VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Systém pro automatizované zpracování papírových formulářů

Automatic Paper Form Handling System

2012

Martin Štěpán

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Štěpán**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Systém pro automatizované zpracování papírových formulářů
Automatic Paper Form Handling System

Zásady pro vypracování:

Testování nabytých znalostí v průběhu nebo závěru probraného učiva je nedílnou součástí výuky. S narůstajícím počtem studentů jsou kladeny stále větší nároky na čas nutný pro opravení velkého množství testů a jejich následného uchování. V poslední době se začínají objevovat systémy pro automatickou opravu testů pomocí prostředků zpracování obrazu. Cílem práce je vytvořit systém pro zadávání testů, jejich automatickou opravu a následné zpřístupnění výsledků pedagogům a studentům.

Ve své práci proveďte:
1. Proveďte rešerši dostupných produktů pro automatické zpracování testů.
2. Na základě získaných informací proveďte implemetaci vlastního detekčního systému.
3. Naimplementujte přehledné webové rozhraní, které bude integrováno se službami VŠB (přihlášení LDAP).
4. Své řešení náležitě otestujte a zhodnoťte.

Seznam doporučené odborné literatury:

[1] Gonzalez, R., C., Woods, R., E.: Digital Image Processing, Prentice Hall, 2002

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Gaura**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 30. dubna 2012 ..... Martin Škěpán .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2012 ..... Martin Škěpán .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Diplomová práce je rozdělena na tři větší celky - první část se zabývá analýzou již existujících systémů pro automatizované zpracování papírových formulářů. V dalších částech práce je popsán postup tvorby vlastního systému.

Druhá část je věnována samotné aplikaci, která se zabývá zpracováním formulářů. Postupně jsou zde popsány metody, které byly použity při tvorbě detektoru. Aplikace je napsána v jazyce C, s využitím knihoven OpenCV a libdecodeqr.

Třetí část se zabývá návrhem interaktivního webového rozhraní, sloužící především pro prezentaci výsledku detektoru. Součástí je i generátor pro tvorbu formulářů.

Klíčová slova: OMR, detekce, formulář, Houghova transformace

Abstract

The thesis is composed from three parts - First part is Analysisof already existing systems for automated processing of paper forms. In the other parts is description for making new original system.

Second part of thesis is dedicated to aplication of system that process the forms. The description of methods used in making of detector is also there. The Aplication is written in C programming language with usage of OpenCV and libdecodeqr libraries.

Third part is about design of interactive web interface, primary serving for presentation of the detector results. The generator for creating forms is included in application.

Keywords: OMR, detection, form, Hough transform

Seznam použitých zkratek a symbolů

BCR	–	Bar Code Reading
ICR	–	Intelligent Character Recognition
MVC	–	Model-View-Controller
OCR	–	Optical Character Recognition
OMR	–	Optical Mark Reading
SSD	–	Sum of squared differences

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Žijeme ve 21. století, v době, kdy se počítače a digitální zpracování dat stávají každodenní součástí našeho života. V dnešní době je cílem především co nejefektivněji získávat a zpracovávat data - jde se cestou digitalizace dat. Blíží se tedy konec papírových formulářů? Zatím to vypadá, že ne. Papírové formuláře stále nacházejí hojné využití všude tam, kde je zatím nelze efektivně nahradit jinou technologií. Velkou nevýhodou těchto formulářů je jejich velká časová a finanční náročnost, neefektivnost při sdílení a také problémy s archivací. Proto se dnes čím dál častěji provádí převod do digitální podoby - tzv. digitalizace dokumentů, která umožňuje mnohem rychleji, efektivněji a přesněji zpracovávat informace obsažené v dokumentech. Klesají také provozní náklady, jelikož není třeba pořizovat kopie a tudíž dochází k nemalé úspoře nákladů.

Máme 2 možnosti jak digitalizovat data:

- Přepis základních údajů z papírového formuláře do předpřipraveného elektronického dokumentu. Nevýhodou je ruční přenesení dat z papíru do počítače a tím vznikající riziko chyby. Další nevýhodou jsou náklady na hardware, software a časová náročnost převodu.
- Naskenování papírových dokumentů. Velkou výhodou je snížení časové náročnosti a minimalizace chyb při převodu dokumentů. Nevýhodou jsou vysoké prvotní náklady, ale i přes tuto nevýhodu se stává tento způsob čím dál více atraktivnější.

Po naskenování dokumentu, tedy převodu do elektronické podoby přichází na řadu rozpoznávání jak tištěných, tak i ručně psaných znaků. Výstupem ze skeneru je grafický soubor, který v podstatě nelze upravit a proto je třeba tento soubor převést do textového formátu. K tomu slouží rozpoznávací programy [1][2].

Mezi nejznámější technologie pro rozpoznávání patří:

- ICR (Intelligent Character Recognition - optické rozpoznávání znaků) - používá pro zpracování ručně vyplněných dokumentů, tedy převodu ručně psaného písma.
- OCR (Optical Character Recognition - inteligentní rozpoznávání znaků) - uplatňuje se při převodu strojově psaných dokumentů.
- BCR (Bar Code Reading - optické rozpoznávání čárových kódů) - převod čárových kódů do počítače (převod do podoby řetězců, číslic a písmen).
- OMR (Optical Mark Reading - optické rozpoznávání značek) - technologie určena pro převod značek v podobě zaškrtaných čtverečků na formuláři do elektronické podoby.

1.1 Struktura práce

Součástí kapitoly 2 je bližší popis technologie OMR a představení aktuálních produktů, komerčních i open source, používaných pro zpracování papírových formulářů.

Kapitola 3 se zabývá návrhem a implementací vlastního systému pro zpracování formulářů. Je zde podrobně popsán postup vývoje této aplikace, včetně popisu všech použitých metod.

Kapitola 4 je věnována webovému rozhraní, sloužící především pro správu, generování a prezentování výsledků z detektoru. Opět je zde podrobně rozebrán návrh aplikace.

Kapitola 5 obsahuje popis průběhu testování.

Kapitola 6 shrnuje výsledky této práce.

2 Technologie OMR

Aplikace, navržená v této práci se řadí mezi OMR systémy a proto je bližšímu popisu této technologie věnována samostatná kapitola.

Tato technologie je označována jako efektivní, rychlá, přesná a cenově nenáročná při zpracování většího množství kategorických dat. Díky těmto výhodám nachází v dnešní době velice široké uplatnění.

Prvním krokem při rozpoznávání je naskenování dokumentu a jeho převod do elektronické podoby. Mnoho OMR systému používá speciální skener, který využívá odrazivosti paprsku světla na předem definovaných místech k detekci označených oblastí. Využívá se toho, že označené oblasti (tedy oblasti se čtverečky) odrážejí méně světla než prázdné plochy papíru.

Druhou možností je použít klasický skener a pro samotnou detekci použít software. Výhodou je, že není třeba žádné speciální zařízení, což umožňuje i běžnému uživateli si vytvořit, vytisknout a následně vyhodnocovat formuláře.

Jak již bylo zmíněno výše, je tato technologie určena pro detekci značek ve formulářích. Nevýhodou je, že formulář musí mít přesně danou strukturu (která je předdefinovaná) - detekují se značky na předem definovaných pozicích. Další nevýhodou (která bývá zároveň označována jako výhoda) je omezení na kategorická data - lze číst jen značky, nelze číst čísla, písmena případně tvar značky. Jediná informace, kterou lze zjistit je, zda je značka vyplněná nebo ne.

Příklad využití OMR systémů: statistické průzkumy, spotřebitelské průzkumy, hodnocení výrobků, testy a hodnocení, loterie a hlasování [3].

2.1 Aktuální produkty pro automatizované zpracování testů

2.1.1 Remark Office OMR

Zástupce komerčního softwaru pro automatizované zpracování a analyzování testů. Jedná se o softwarový balík v platformě windows vytvořený firmou Gravic. Dokáže rozpoznat několik druhů optických značek, strojově tištěný kód a čárové kódy.

Vytvářet formuláře lze v jakémkoliv textovém editoru. Uživatel má tedy plnou volnost při vytváření a designování formulářů.

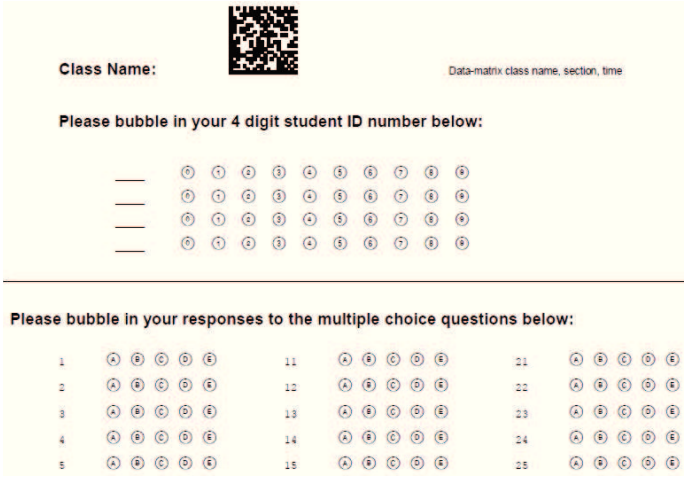
Velká výhoda tohoto softwaru spočívá v tom, že nevyžaduje, aby se ve formuláři nacházely speciální značky. Díky tomu není uživatel prakticky omezen v umístění optických značek. Jak již bylo zmíněno v předchozí kapitole, před naskenováním formuláře je třeba, aby byla definována šablona - prostorové uspořádání značek. K tomuto slouží část programu - Remark Office OMR Template Editor. Uživatel nejprve naskenuje formulář a poté pomocí myši specifikuje oblast se značkami, kterou ještě dále definuje (např. se může jednat o název oblasti nebo názvy datových proměnných).

Pro naskenování formulářů není třeba žádný speciální skener a lze tedy využít i úplně obyčejný. Jak bylo zmíněno výše, tak software zpracovává optické značky a čárové kódy. Pro zpracování strojově tištěného kódu lze využít buď interní OCR rozpoznávač nebo

využít jakýkoliv jiný OCR systém. Remark Office OMR obsahuje systém pro správu výjimek, který upozorňuje na nekorektní informace a dovoluje je uživateli opravit.

Proces analýzy je automatizován, například v případě známkování testů je třeba jen specifikovat správné odpovědi, bodování a specifikovat nastavení a o vše ostatní se už postará software. Systém také poskytuje statistiky a grafy analyzovaných položek a nabízí také report analýzy testu, s výslednou známkou, úspěšností a zvýrazněním špatných odpovědí.

Naskenované a zpracované formuláře mohou být exportovány do různých datových formátů (např. Acces, Excel). Software dovoluje i uložení do formátu HTML [4][5].



Obrázek 1: Ukázka části formuláře v Remark Office [4]

2.1.2 Addmen OMR Pro

Komerční softwarový balík vytvořen firmou Addmen.

Tento balík má v sobě integrovaný Question Paper Generator modul, který je určen pro generování formulářů s otázkami. Lze buď využit předdefinované šablony nebo si vytvořit vlastní. Umožňuje zadávat různé typy otázek: otázky s více než jednou odpovědí, ano/ne otázky a otázky jejichž výsledkem jsou číselné hodnoty (celá čísla).

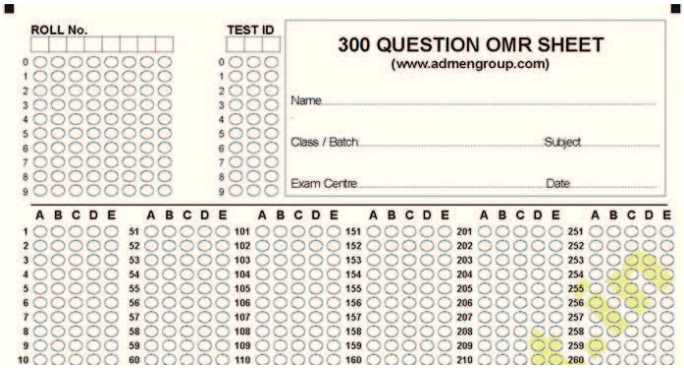
Po analýze a zpracování formulářů jsou k dispozici detaily jednotlivých formulářů - korektně, nekorektně a neoznačené odpovědi a různé statistiky.

Výsledky je možno exportovat do různých formátů - např. MS Access, MS Excess, Word, HTML [6].

2.1.3 Vision OMR

Další zástupce komerčního produktu.

Vision OMR je softwarový balík určený pro platformu windows. Umožňuje rozpoznávat optické značky kruhových nebo čtvercových tvarů a čárové kódy.



Obrázek 2: Ukázka části formuláře v Addmen OMR Pro [6]

Vytvořit šablonu formuláře lze v jakémkoliv textovém editoru. Unikátnost tohoto softwaru spočívá v tom, že nejsou potřeba žádné speciální optické markery, takže má uživatel naprostou volnost při tvorbě dotazníku. Před samotným naskenováním a zpracováním je nejprve třeba absolvovat trénovací proces - specifikovat šablonu formuláře. K tomu slouží modul OMR Template Editor. Jednoduchou procedurou se v naskenovaném formuláři specifikují oblasti s optickými značkami a vytvoří se šablona pro detekci.

Po naskenování následuje analýza, dle předem vytvořené šablony a vyhodnocení formuláře. Součástí vyhodnocení jsou i různé statistiky.

Výsledky lze exportovat do více než 30-ti různých výstupních formátů - např. Access, Excel nebo Lotus 1-2-3. Je zde také možnost uložit data do formátu HTML [7].

2.1.4 TeleForm

Zástupce komerčního produktu, vytvořeného firmou Cardiff.

Opět se jedná o kompletní softwarový balík, který dokáže rozpoznávat ručně psané znaky (ICR), strojově tištěný kód (OCR), optické značky (OMR) a čárové kódy.

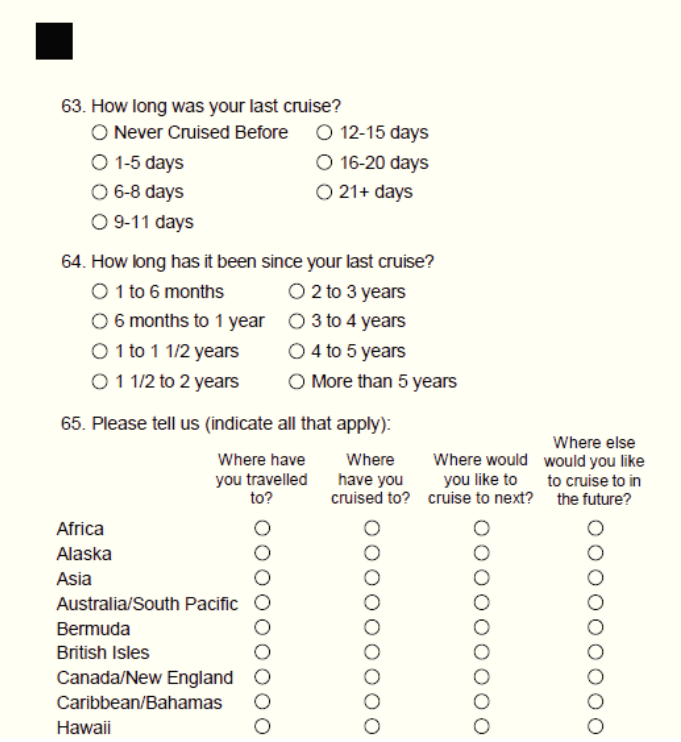
Pro tvorbu formulářů je určen modul Designer Module. Vytváří se zde kompletní design, ale také atributy polí a validace.

Dalším modulem je Scan Station, určený pro načtení nastavení formulářů (např. se může jednat o druh formuláře) a nastavení skeneru. Po naskenování přichází na řadu analýza pomocí Reader modulu. Pomocí něj se identifikuje formulář a provede první validace.

Posledním krokem je verifikace. K tomu je určen Verifier Module. Používá se pro přezkoumání formulářů, u kterých selhala validace a vyžadují ruční validaci od uživatele [8].

2.1.5 queXF

Zástupce open source OMR systému, vyvíjen společností Australian Consortium for Social and Political Research Incorporated (ACSPRI).



63. How long was your last cruise?

○ Never Cruised Before	○ 12-15 days
○ 1-5 days	○ 16-20 days
○ 6-8 days	○ 21+ days
○ 9-11 days	

64. How long has it been since your last cruise?

○ 1 to 6 months	○ 2 to 3 years
○ 6 months to 1 year	○ 3 to 4 years
○ 1 to 1 1/2 years	○ 4 to 5 years
○ 1 1/2 to 2 years	○ More than 5 years

65. Please tell us (indicate all that apply):

	Where have you travelled to?	Where have you cruised to?	Where would you like to cruise to next?	Where else would you like to cruise to in the future?
Africa	○	○	○	○
Alaska	○	○	○	○
Asia	○	○	○	○
Australia/South Pacific	○	○	○	○
Bermuda	○	○	○	○
British Isles	○	○	○	○
Canada/New England	○	○	○	○
Caribbean/Bahamas	○	○	○	○
Hawaii	○	○	○	○

Obrázek 3: Ukázka části formuláře v TeleForm [9]

Primárně je tento software určen pro detekci optických značek, ale může být také využit při detekci ručně psaných znaků a čísel.

Pro tvorbu formulářů se využívá program queXML, který je součástí queX balíku. Výhodou tohoto softwaru je, že ověřování a potvrzování správnosti detekce je přes webové rozhraní a tedy není nutné instalovat další pomocný software. Další výhodou je automatická detekce boxů ve formuláři (není třeba ručně specifikovat pozici boxů) [9].

2.1.6 Shared Questionnaire System(SQS)

Další z řady open source zástupců. Jedná se o OMR systém pro zpracování formulářů založených na XML standartech. Software je implementován v jazyce Java, XSLT a JavaScript. Je tedy multiplatformní a lze ho jednoduše nainstalovat a spustit z webového prohlížeče pomocí JavaWebStart [10].

2.1.7 Další produkty

Existuje velké množství jak komerčních, tak volně dostupných produktů.

Z komerčních lze například ještě zmínit TestsChecker nebo FormReturn a z volně dostupných například Udai OMR.

3 Detekční systém

Jak již bylo zmíněno v úvodu, tato kapitola se zabývá popisem návrhu a implementace detekčního systému, určeného pro zpracování testových formulářů - zjištění odpovědí na jednotlivé otázky. Pro lepší pochopení činnosti detektoru jsou před popisem vlastní implementace vysvětleny některé důležité pojmy a použité algoritmy při realizaci aplikace.

3.1 Knihovna OpenCV

OpenCV je otevřená multiplatformní knihovna určená především pro zpracování obrazu v reálném čase. Původně byla vyvinuta společností Intel, ale nyní je knihovna volně šiřitelná pod licencí BSD. Využití je především v programovacích jazycích C a C++, ale s generátorem rozhraní SWIG ji lze použít také v Pythonu [11].

3.2 Histogram

Histogram je statistická veličina, která popisuje pravděpodobnost výskytu úrovně jasu v obraze. Vyjadřuje se jako graf, na jehož vodorovnou osu se vynáší jas a na svislou osu počet pixelů, odpovídajících danému jasu. Celkový počet buněk odpovídá ploše obrázku. Příklad histogramu lze vidět na obrázku (4).

Obrázek 4: Obrázek a jeho histogram [12]

3.3 Konvoluce

Jedná se o matematický operátor, jehož vstupem jsou dvě funkce - f a g. Výsledná hodnota v bodě (x, y) se vypočítá jako vážený průměr funkce f s váhovou funkcí g. Ve zpracování obrazu typicky bývá funkce f vstupní obraz a funkce g filtr (často nazýván konvoluční filtr, respektive konvoluční maska) [13].

Diskrétní 2D konvoluce je dána vztahem (1):

$$f(x,y) * g(x,y) = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f(i,j)g(x-i, y-j) \tag{1}$$

3.4 Detekce rohů

V aplikaci se využívá detekce rohů pro nalezení rohů jednotlivých markerů. Pro zlepšení výkonu probíhá hledání rohů nad obrázkem, zpracovaným detektorem hran.

3.4.1 Moravec

Jedná se o jeden z nejstarších detektorů rohů. Pracuje na principu zkoumání každého pixelu obrazu a podobnosti tohoto bodu s jeho okolím. Nejprve se nad vybraný pixel umístí malá čtvercová oblast (typicky 3x3, 5x5 nebo 7x7 bodů). Pak následuje pohyb po okolí - posun o 1 pixel horizontálně, vertikálně a diagonálně. Opět nad daný pixel umístíme čtverec a počítáme podobnost (jasovou odlišnost). Ta se vypočítá jako suma rozdílu čtverců nad vybraným pixelem a pixelem z jeho okolí. Čím menší bude suma, tím větší bude podobnost zkoumaných pixelů. Bod je označen jako rohový (respektive významný) pokud není podobný svému okolí. Tedy že se jeho jas významně mění ve všech zkoumaných směrech. Při rozhodování o rohovosti se počítá s minimální hodnotou jasu, nalezenou při zkoumání okolí.

Tento algoritmus má jednu nevýhodu - pokud se hrana nenachází ve vertikální, horizontálním nebo diagonálním směru, pak nebude detekována jako významný bod [14][16].

3.4.2 Harris and Stephens

Harris a Stephens vychází z detektoru publikovaného Moravcem. Jedná se o vylepšení tohoto detektoru - lepší detekce a stabilita, ale za cenu vyšších výpočetních nároků. Liší se použitím takzvané lokální autokorelační funkce.

Výhodou Harrisova detektoru je nezávislost na posunu, rotaci a odolnost vůči šumu. Nevýhodou tohoto algoritmu je, že se nedokáže vypořádat se změnou měřítka.

Metoda je stejně jako Moravcova, založena na posouvání snímku ve všech základních směrech. Rozdíl je, že Harris bere ohled na směr posunutí snímku.

Základním principem tohoto algoritmu je výpočet rozdílu čtverců jasů (SSD) v každém bodě. Velkým omezením ale je, že porovnávané snímky musí mít stejný kontrast a jas. Pro každý se vypočítá autokorelační matice.

Předpokládejme na vstupu 2-rozměrný šedotónnový obraz, reprezentován jasovou funkcí I. Rozdíl čtverců jasů S se vypočte dle následujícího vzorce (2):

$$S(x,y) = \sum_u \sum_v w(u,v)(I(u+x, v+y) - I(u,v))^2 \tag{2}$$

kde (x, y) značí posun a (u, v) okolí bodu.
$I(u + x, v + y)$ lze aproximovat Taylorovým polynomem (3):

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y \tag{3}$$

kde I_x a I_y jsou parciální derivace I.
Výsledkem je aproximace (4):

$$S(x,y) \approx \sum_u \sum_v w(u,v)(I_x(u,v)x + I_y(u,v)y)^2 \tag{4}$$

Přepisem do maticového tvaru dostaneme (5):

$$S(x,y) \approx (x \quad y) A \begin{pmatrix} x \\ y \end{pmatrix} \tag{5}$$

kde A znamená (6):

$$S(x,y) = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} < I_x^2 > & < I_x I_y > \\ < I_x I_y > & < I_y^2 > \end{bmatrix} \tag{6}$$

Matice A se nazývá Harrisova matice. Z této matice vypočítáme vlastní čísla, na základě kterých určíme zda bod patří mezi významné nebo ne.
Vlastní čísla se vypočítají dle následujícího vzorce (7):

$$\lambda_{1,2} = \frac{< I_x^2 > + < I_y^2 > \pm \sqrt{(< I_x^2 > - < I_y^2 >)^2 + 4 < I_x I_y >< I_x I_y >}}{2} \tag{7}$$

a platí pro ně následující pravidla:

- Pokud $\lambda_1 \approx 0 \wedge \lambda_2 \approx 0$, pak nebyl nalezen žádný významný bod.
- Pokud $\lambda_1 \approx 0 \wedge \lambda_2$ je velké kladné číslo, pak byla nalezena hrana.
- Pokud λ_1 i λ_2 jsou velká kladná čísla, pak byl nalezen roh.

Výpočet vlastních čísel je výpočetně náročný (z důvodu počítání odmocniny) a proto Harris a Stephens navrhli vzorec, na základě kterého se rozhoduje, zda je v bodě hrana nebo ne (8):

$$M_c = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = det(A) - \kappa Trace^2(A) \tag{8}$$

Hodnota κ byla určena experimentálně a uvádí se přijatelné rozmezí od 0,04 do 0,15. Před použitím Harrisova detektoru je doporučeno odstranit šum pomocí Gaussova filtru [14][15][17][18].

3.4.3 The Wang and Brady corner detection algorithm

Tento algoritmus je navržen na jiném principu, než předchozí dva. Považuje vstupní obrázek za křivku a hledá místa, kde dochází k náhlé změně směru křivky (velkému zakřivení).

Měřítko zakřivení C se vypočítá následovně (9):

$$C = \nabla^2 I - c|\nabla I|^2 \tag{9}$$

kde c označuje citlivost detektoru.

K redukci šumu se zde využívá vyhlazování (doporučen je Gaussián). Po aplikování vyhlazení dojde k posunutí rohů a je tedy třeba ještě aplikovat korekční faktor, aby byly výsledky korektní [14].

3.4.4 The SUSAN corner detector

Princip tohoto detektoru spočívá v umístění kruhové masky nad testovaný pixel. Poté se porovnává hodnota jasu testovaného pixelu s hodnoty jasu v kruhovém okolí, dle porovnávací funkce (10):

$$c(\vec{m}) = e^{-(\frac{I(\vec{m})-I(\vec{m_0})}{t})^6} \tag{10}$$

kde t označuje poloměr, $\vec{m_0}$ testovaný pixel a $\vec{m}$ pixel masky, přičemž platí, že $\vec{m} \in M$, kde M představuje plochu masky. Exponent se stanovuje empiricky. Postupně se prochází celý vstupní obrázek a tento postup se aplikuje na každý bod.

Oblast SUSAN je dána vztahem (11):

$$n(M) = \sum_{\vec{m} \in M} c(\vec{m}) \tag{11}$$

jestliže je c kruhová funkce, pak n značí počet pixelů v masce, které jsou ve vzdálenosti t od jádra masky (testovaného pixelu). SUSAN operátor lze vypočítat podle následujícího vztahu (12):

$$f(n) = \begin{cases} g - n(M) & \text{if } n(M) < g \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

kde g značí geometrickou prahovou hodnotu. Jinak řečeno, výsledek SUSAN operátoru je kladný pouze pokud je plocha dostatečně malá. Nejmenší SUSAN oblasti lze nalézt potlačením nemaximálních hodnot. Výsledkem je pak SUSAN operátor.

Pro detekci rohů je potřeba ještě provést 2 kroky. Prvním je nalezení těžiště SUSAN oblasti - roh má těžiště daleko od středu masky. Druhým krokem je ověření, že všechny body, které leží na přímce spojující střed a těžiště, až ke kraji masky, jsou v SUSAN oblasti.

Výhodou této metody je, že při posouvání kamery nemusí docházet k multidetekci rohů - opětovnému detekování již nalezených rohů [14].

3.4.5 The Trajkovic and Hedley corner detector

Tento algoritmus pracuje podobným způsobem jako výše zmíněný SUSAN algoritmus. Opět se pracuje s okolím pixelu, v podobě malého kruhu a všemi úsečkami, které prochází středním pixelem a končí na hranici okolí.

Střed kruhového okolí označíme S. Po proložení přímky tímto středem vzniknou na průsečíku kruhového okolí a přímky body P a P'. Hodnota rohovosti se vypočítá dle vztahu (13):

$$C(x,y) = min((I_P - I_S)^2 + (I_{P'} - I_S)^2) \tag{13}$$

kde I označuje intenzitu v daném bodě.

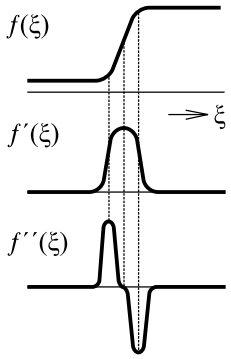
Pokud vyjde rohovost nízká, pak existuje nejméně jedna úsečka, pro kterou platí, že intenzita I_S je přibližně rovna intenzitám I_P a $I_{P'}$. A kruhové okno tedy leží uvnitř homogenního objektu.

V případě, kdy je střed okna nad rohem objektu, tak na každé úsečce je alespoň jeden z bodů I_P, $I_{P'}$ mimo objekt a tedy má jinou hodnotu než I_S. Proto bude rohovost vysoká.

Speciálním případem je, když existuje právě jedna úsečka, na které je intenzita středu I_S přibližně rovna intenzitám I_P, $I_{P'}$. V tomto případě se jedná o hranu [14][15].

3.5 Detekce hran

Hrana je místo v obraze, kde dochází k náhlé změně jasu - v místě hrany je velká hodnota derivace jasové funkce (5).



Obrázek 5: Jas a jeho derivace [13]

Základní rozdělení hranových detektorů, na základě technik použitých k detekci hran: hledání maxima v první derivaci, hledání průchodu druhé derivace nulou a parametrické modely hran.

Níže uvedené detektory patří do první zmíněné kategorie - používají první derivaci pro hledání hran [13].

3.5.1 Roberts

Jeden z nejstarších a dnes už málo používaných detektorů. Pro detekci hran se využívá okolí 2x2 pixely.

Robertsův operátor využívá následující masky (14):

$$h_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, h_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{14}$$

Velkou nevýhodou tohoto operátoru je malá velikost masky (derivace v jediném bodě) - díky tomu je i šum detekován jako hrana, proto nelze tento operátor použít na zašumělé obrázky [13][19].

3.5.2 Operátor Prewittové

Tento operátor používá pro detekci hran hodnoty obrazové funkce ze všech sousedních pixelů (maska 3x3 pixely). Výsledná hodnota se pak stanoví jako průměr. Operátor Prewittové používá následující masky (15) [13][19]:

$$h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, h_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{15}$$

3.5.3 Sobel

Tento operátor se velice podobá operátoru Prewittové. Rozdíl je, že Sobel dává větší váhu středu a výsledek počítá jako vážený průměr. Sobelův operátor používá následující masky (16) [13][19]:

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, h_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} ... \tag{16}$$

3.5.4 Canny

Tento algoritmus navrhl v roce 1986 John F. Canny. Je často označován jako optimální hranový detektor. Při návrhu si Canny stanovil 3 základní požadavky, které by měl tento detektor splňovat. Nalezení detektoru pak představovalo optimalizační úlohu. Hlavní kritéria Cannyho detektoru hran: minimalizace chybné detekce, co nejpřesněji lokalizovat hrany a jednoznačná identifikace bodů hrany.

Jednotlivé kroky Cannyho detektoru

1. Eliminace šumu - většinou používá se Gaussův filtr. Dvojrozměrná varianta Gaussova normálního rozdělení je definována jako (17):

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{17}$$

kde x, y označují souřadnice bodu v obraze. σ je standartní odchylka rozdělení. Výsledkem vzorce je konvoluční maska, která je pak aplikována na zadaný obraz.

2. Výpočet gradientu - nejčastěji se využívá Sobelův operátor. Výsledkem je nejen velikost gradientu, ale i jeho směr, který je potřebný pro další výpočet.

3. Ztenčení - z hodnot gradientů odebereme body, které nejsou maximem - necháme jen lokální maxima. Výsledkem je ztenčení hran a získáme jednoznačnou identifikaci hrany.

4. Prahování - výsledkem předchozího kroku je určení hran a jejich polohy. Detekovány jsou ale všechny hrany, tedy i velice malé, které nás nezajímají. Proto se stanovuje minimální a maximální hodnota, dle kterých se stanoví významnost hrany. Pokud hodnota gradientu je větší než maximální hodnota, pak je bod označen jako hranový. V případě, že je mezi minimální a maximální hodnotou, pak je označen jako hranový jen v případě, že sousedí s bodem, který byl označen jako hranový [13][19][20].

Obrázek 6: Ukázka Cannyho detektoru [19]

3.6 Geometrické transformace

Geometrické afinní transformace patří mezi nejčastěji používané operace v počítačové grafice. Afinní transformace je dána vztahem (18):

$$P' = P \cdot A \tag{18}$$

kde $P = [x, y, w]$ je bod, který se transformuje maticí A na bod $P' = [x', y', w']$.
Mezi afinní transformace patří: posunutí, rotace, změna měřítka, zkosení a kombinace výše uvedených.

3.6.1 Posunutí

Jedná se o posun bodu $P = [x, y]$ o vektor $p = (X_T, Y_T)$. Výsledkem je bod P' (19)(20):

$$X' = X + X_T \tag{19}$$
$$Y' = Y + Y_T \tag{20}$$

Po přepisu do maticového tvaru a převodu do homogenních souřadnic dostaneme (21):

$$A_T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ X_T & Y_T & 1 \end{bmatrix} \tag{21}$$

3.6.2 Rotace

Výsledkem rotace bodu $P = [x, y]$ kolem počátku souřadného systému $O = [0, 0]$ o orientovaný úhel α je bod P' (22)(23):

$$X' = X \cdot \cos\alpha - Y \cdot \sin\alpha \tag{22}$$
$$Y' = X \cdot \sin\alpha + Y \cdot \cos\alpha \tag{23}$$

Po přepisu do maticového tvaru a převodu do homogenních souřadnic dostaneme (24):

$$A_R = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{24}$$

3.6.3 Změna měřítka

Změnou měřítka se provede změna velikosti objektu ve směru souřadných os. Jedná se buď o zmenšení (pokud je koeficient v rozsahu 0-1) nebo zvětšení (pokud je koeficient větší než 1). Znaménko ovlivňuje směr zvětšení (případně zmenšení).

Výsledkem změny měřítka bodu $P = [x, y]$ dostaneme (25)(26):

$$X' = S_X \cdot X \tag{25}$$
$$Y' = S_Y \cdot Y \tag{26}$$

kde S_X označuje změnu měřítka na ose x a S_Y na ose y.

Po přepisu do maticového tvaru a převodu do homogenních souřadnic dostaneme (27):

$$A_S = \begin{bmatrix} S_X & 0 & 0 \\ 0 & S_Y & 0 \\ X_T & Y_T & 1 \end{bmatrix} \tag{27}$$

3.6.4 Skládání transformačních matic

Složením více transformací můžeme provést jednu velkou transformaci místo několika menších. Výslednou matici získáme postupným násobením dílčích matic, reprezentující jednotlivé transformace. Je ale třeba dávat velký pozor na pořadí transformací, respektive na pořadí násobení matic. Pokud bod nejprve posunem a pak s ním zarotujeme, tak nedostaneme stejný výsledek, jako v případě rotace a až pak posunu. Používá se zápis $P' = P \cdot A$ a proto musíme jednotlivé matice násobit zprava.

3.7 Houghova transformace

Houghova transformace nachází velice široké uplatnění ve zpracování obrazu. Využívá se pro nalezení různých geometrických útvarů. Omezením je, že útvary, které chceme vyhledávat, musí být popsány parametricky. Proto se tato metoda používá pro detekce jednoduchých útvarů jako jsou přímky, kružnice, elipsy, čtverce a podobně.

Vstupní obraz obvykle prochází předzpracováním - například se využívá detekce hran (nejčastěji Cannyho detektor). Díky tomu se zvyšuje šance, že bude nalezena požadovaná struktura a zároveň se značně snižuje časová náročnost celého algoritmu.

Výstupem jsou nalezené útvary v parametrické formě.

3.7.1 Detekce čar

Jak již bylo zmíněno výše, potřebujeme čáru (respektive přímku) vyjádřit v parametrickém tvaru. V kartézské soustavě lze přímku parametricky vyjádřit následovně (28):

$$y = a \cdot x + b \tag{28}$$

Problémem zde jsou přímky rovnoběžné s osou Y. Pro tyto přímky nabývá a hodnoty nekonečno a proto toto vyjádření není vhodné.

Mnohem vhodnější je vyjádřit přímku v polárních souřadnicích (29):

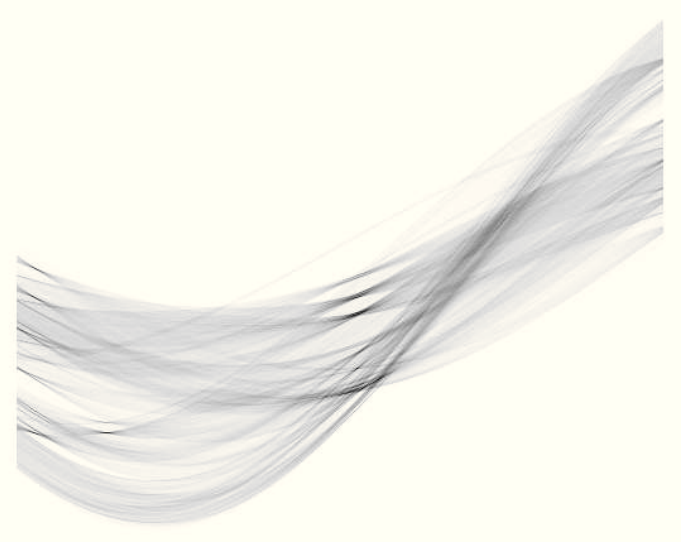
$$r = x \cdot cos\theta + y \cdot sin\theta \tag{29}$$

kde r označuje kolmou vzdálenost přímky od počátku a θ je úhel, který svírá přímka s osou X.

Pro detekci žádaných útvarů se používá tzv. Houghův prostor. Jedná se o 2D graf, kde se na vodorovnou osu vynáší θ a na svislou r.

Základním principem při detekci je výpočet hodnot θ a r v každém bodě obrazu. Výpočet spočívá v dosazení (x, y) souřadnic do rovnice (29). Množina všech možných řešení pro zadaný bod vytvoří v Houghově prostoru spojitou křivku. Jestliže výpočet

opakujeme pro všechny body, získáme množinu křivek odpovídajícím jednotlivým bodům obrazu. Křivky se v určitých bodech protínají - tyto body tvoří maxima v akumulační rovině (r_{max}, θ_{max}) a určují parametry hledaných přímek v obraze [21][22].



Obrázek 7: Křivky v Houghově prostoru [23]

3.7.2 Detekce kružnic

Parametrické vyjádření kružnice je dáno vzorcem (30):

$$(x-a)^2 + (y-b)^2 = r^2 \tag{30}$$

kde a a b označují souřadnice středu kružnice a r znamená poloměr kružnice.
Princip je stejný jako u detekce čar - opět je třeba najít parametry z rovnice (30). Počet parametrů se zvedl na 3 a to znamená, že Houghův prostor bude nyní trojrozměrný. Díky tomu hodně naroste časová i paměťová složitost celé operace.

Problémem při detekci kružnic jsou kružnice, které mají mnohem větší poloměr než jsou rozměry obrázku. Nejen rozměr r v Houghově prostoru by musel být dostatečně velký, ale i zbylé 2 rozměry, protože střed kružnice může ležet mimo rozměry obrázku. Vznikl by obrovský trojrozměrný Houghův prostor, který by ztratil všechny výhody Houghovi transformace. Proto se poloměr kružnic omezuje na minimální a maximální hodnotu (omezením poloměru se zároveň omezí i zbylé parametry) a tím se velice sníží jak časová, tak i paměťová složitost.

Samotná detekce se od detekce přímek trošku liší. Souřadnice každého vstupního bodu dosadíme do rovnice (30), místo parametrů a a b. Poté postupně dosazujeme poloměr r (který se postupně inkrementuje o předem definovanou hodnotu) a počítáme hodnoty x a y. V Houghově prostoru pak na těchto souřadnicích (x,y) do aktuální hloubky r zvýšíme intenzitu (inkrementujeme hodnotu).

V místě průsečíku objektů vznikají maxima, která představují hledané parametry [21].

3.8 QR kód

V dnešní době se s QR kódem začínáme setkávat čím dál častěji - na internetu, v časopisech, na plakátech atd. Jedná se o kód podobný čárovému kódu, který všichni dobře známe. U čárového kódu se jedná o obrázek, v kterém jsou informace zakódovány pomocí číslic, které lze strojově přečíst a získat požadovanou informaci. Čárový kód pojme maximálně 12 cifer.

QR kód byl vytvořen japonskou společností Denso-Wave v roce 1994. Zkratka QR znamená Quick Responce - vyjadřuje, k čemu je kód určen. K rychlému načtení a dekódování informací. Původně tento systém sloužil pro sledování pohybu výrobku v továrně, ale v dnešní době má velice široké uplatnění všude, kde je třeba rychle předat větší množství informací.

Způsob kódování je odlišný oproti čárovým kódům. V QR kódu je informace zakódována pomocí bílých a černých čtverečků, uspořádaných do matice. Velikost je různá, závislá na množství informací, které chceme uložit. Maximální množství znaků, které lze uložit je 4300. Uložit lde libovolná textová data - nejběžněji se ukládají internetové adresy, kontaktní informace, e-mail, telefonní číslo atd. QR kód je složen z několika částí: terčíky pro zaměření při čtení, formátu kódu, verzi kódu a samotných dat.

Obrázek 8: Příklad QR kódu - zakódování URL www.vsb.cz

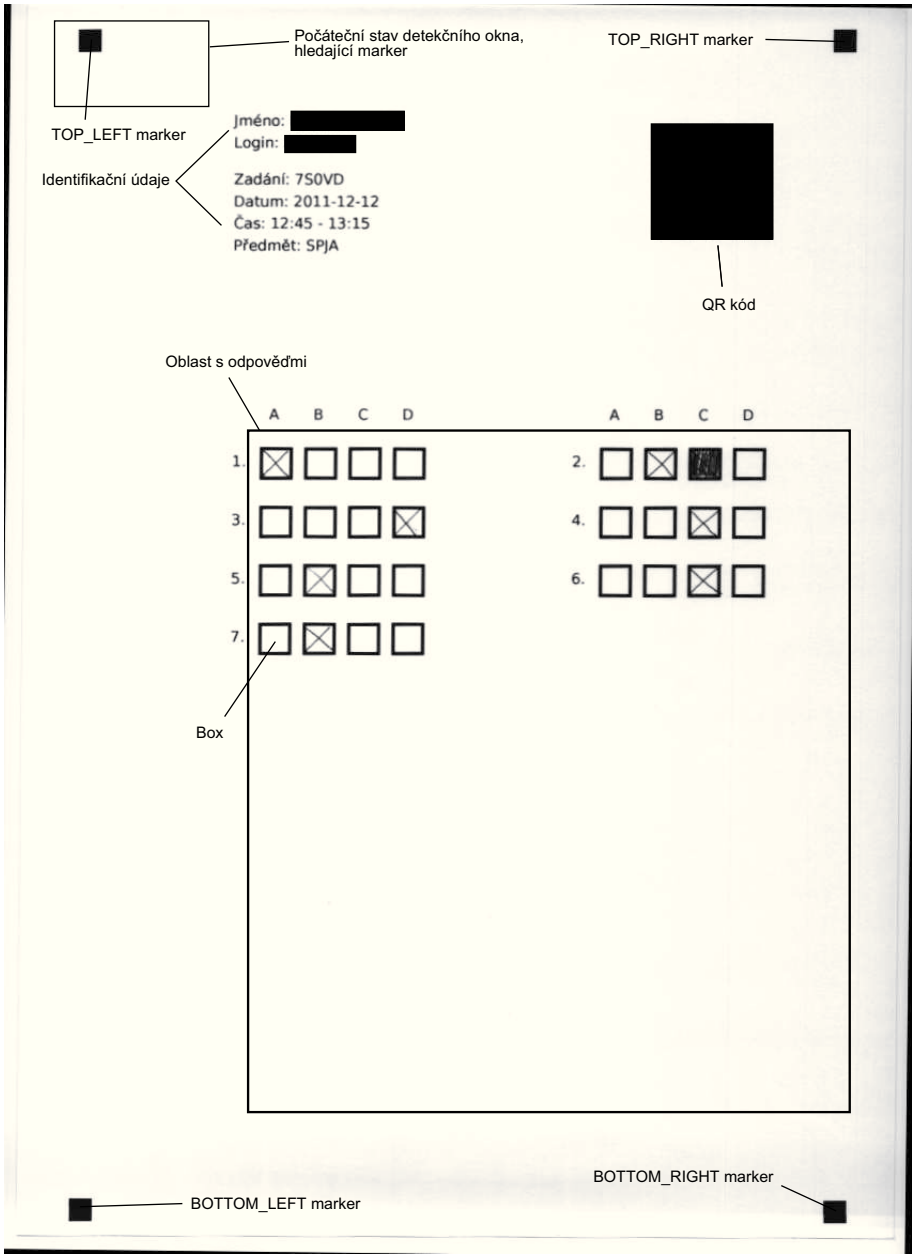
3.9 Popis implementace vlastního detektoru

3.9.1 Pomocné makra a vlastní datové typy

Pro potřeby implementace a především pro zpřehlednění a efektivnější práci bylo vytvořeno několik maker a pomocných struktur. Pro lepší orientaci v zavedených pojmech je zde přiložen obrázek se znázorněním základních prvků (9).

Pomocné struktury:

- Line - struktura definující úsečku, definovanou dvěma body.



Obrázek 9: Test se znázorněním základních prvků

- Lines - struktura reprezentující pole úseček. Obsahuje počet úseček a samotné pole úseček, reprezentovaných datovým typem Line.
- Answer - struktura pro uložení odpovědi. Ukládá se číslo otázky, samotná odpověď (reprezentována celým číslem) a souřadnice levého horního bodu boxu s odpovědí.

- Result - struktura určená pro uložení výsledků detektoru. Obsahuje počet otázek, pole s odpověďmi a pole obsahující souřadnice levého horního bodu všech nalezených boxů (tedy jak nezaškrtnutých, tak zaškrtnutých a začerněných).
- Marker - struktura reprezentující marker, který je reprezentován levým horním a pravým dolním bodem.
- Settings - struktura pro uložení informací načtených z konfiguračního souboru. Jedná se o: rozměr boxu, horizontální vzdálenost dvou boxů, vertikální vzdálenost dvou boxů, horizontální vzdálenost prvního boxu první otázky a prvního boxu druhé otázky, limit, určující rozdíl počtu pixelů mezi nezaškrtnutým a zaškrtnutým boxem, limit (zadán jako hodnota v procentech), od kterého se box bere jako začerněný, počet možností odpovědi na jednu otázku, definování x-ové a y-ové souřadnice odsazení detekčního okna hledající markery od okraje obrázku (x1-offset a y1-offset) a definování x-ového a y-ového offsetu pro nastavení oblasti s odpověďmi (x2-offset a y2-offset). Odsazuje se nepočítá od začátku obrázku, ale od konce nalezených markerů.
- Box - struktura reprezentující box. Obsahuje informaci o počtu bílých pixelů a souřadnici levého horního bodu boxu.
- Login - struktura obsahující login studenta. Vnitřně uloženo jako statické pole znaků a počet znaků loginu.
- Version - stejné jako struktura login, akorát se jedná o verzi zadání.
- Date - stejné jako struktura login, s rozdílem že je zde uloženo datum zadání testu.
- Subject - stejné jako struktura login, obsahující informaci z jakého předmětu je daný test.

3.9.2 Načtení konfigurace

Prvním krokem je načtení konfiguračního souboru pomocí funkce *LoadSettings(char *filename)*, která jako parametr požaduje jméno konfiguračního souboru. Funkce vrací strukturu Settings, obsahující nastavení pro detektor. Pokud není soubor nalezen nebo nastane chyba při jeho otvírání, tak se vypíše chybová hláška a aplikace se ukončí. Jednotlivé položky této struktury jsou detailněji popsány v kapitole 3.9.1. V této aplikaci se konfigurace načítá ze souboru config.txt.

3.9.3 Načtení obrázku

Dalším krokem je načtení obrázku, na kterém bude probíhat detekce. K tomu slouží OpenCV funkce *cvLoadImage(const char* filename, int iscolor=CV_LOAD_IMAGE_COLOR)*. Tato funkce vrací ukazatel na obrázek v případě úspěšného načtení a NULL v případě neúspěchu. Jméno obrázku se načíná jako parametr příkazové řádky. V případě, že obrázek není nalezen nebo ho nelze otevřít, se vypíše oznámení a program se ukončí.

3.9.4 Dekódování QR kódu

Pro dekódování QR kódu se využívá externí knihovna libDecodeQR. Výstupem z této knihovny je řetězec ve tvaru: login;zadani;datum;predmet. Pro zjištění jednotlivých informací jsou určené funkce: *GetLogin(char* str)*, *GetVersion(char* str)*, *GetDate(char* str)* a *GetSubject(char* str)*. Všechny požadují jako vstup řetězec definovaný výše a vrací strukturu, obsahující řetězec s danou informací (tedy login, verzi zadání, datum nebo čas) a délku tohoto řetězce. Parser využívá toho, že informace jsou odděleny středníkem a na základě tohoto se ze vstupního řetězce vytáhne potřebné informace. Pokud nejsou nalezene všechny informace, tak se vypíše chyba a detekce se ukončí.

3.9.5 Detekce markerů

Dalším krokem je detekce markerů, na základě kterých dochází k vyrovnání obrázku a detekce oblasti s odpověďmi. Slouží k tomu funkce *DetectMarkers(IplImage* src, Settings s)*, která vrací 4-prvkové pole datového typu Marker. Pokud nejsou nalezeny všechny 4 markery, tak funkce vrací NULL. Detekce každého markeru probíhá postupně - nejprve se detekuje levý horní, pak pravý horní atd. Detekce probíhá v cyklu, dokud není nalezen marker nebo není překročen stanovený limit počtu cyklů. Na začátku se vytvoří malé detekční okno, které je odsazeno od počátku obrázku dle zadaných parametrů a postupně prochází spirálovitě obrázek a hledá marker.

K samotné detekci slouží funkce *DetectMarker(IplImage* src, int position, CvPoint roiCenter, int roiW, int roiH)*. Parametry této funkce jsou: vstupní obrázek, pozice markeru, který chceme detekovat (pro definici pozice slouží makra popsána v kapitole 3.9.1) a posledním parametrem je oblast ve které se hledá (zadaná středem obdélníku a jeho rozměry).

Pro zlepšení detekce je vstupem funkce obrázek zpracovaný Cannyho detektorem hran. Na tento vstup se aplikuje Harrisův detektor rohů. Pokud je počet rohů jiný než 4, pak funkce vrací marker se souřadnicemi -1 - oznámení, že marker nebyl nalezen. Pokud je marker nalezen, tak se vrací nalezené souřadnice.

3.9.6 Vyrovnání obrázku

Po úspěšném nalezení markeru se provádí vyrovnání obrázku, pomocí funkce *EqualizationImage(IplImage* src, Marker *markers)*. Tato funkce vrací vyrovnaný obrázek. Princip spočívá ve výpočtu úhlu rotace na základě detekovaných markerů, pomocí funkce *CalculateAngle(Marker *markers)*, která vrací úhel v radiánech. Pro výpočet úhlu se vezme levý horní bod levého horního a pravého horního markeru. Vytvoří se přímka, jejichž směrnice dává úhel, o který je třeba s obrázkem zarotovat. Vlastní rotaci provádí funkce *RotateImage(IplImage* image, float angle)*, která aplikuje afinní transformaci (rotaci) na zadaný obrázek. Výstupem je vyrovnaný (rotovaný) obrázek.

3.9.7 Přepočet a označení markerů

Pro přepočet souřadnic markerů slouží funkce *RecalculateMarkers(Marker *markers, int imgW, int imgH, float angle)*. Jako parametry požaduje pole markerů, rozměry obrázku a úhel rotace, který zjistíme stejně jako při vyrovnávání obrázku - funkcí *CalculateAngle(Marker *markers)*. Funkce *RecalculateMarkers* přepočítáva každý bod markeru. Pro tuto potřebu byla vytvořena funkce *RecalculatePoint(CvPoint p, float angle, int w, int h)*. Která požaduje bod, úhel a rozměry obrázku. Výsledkem je přepočítaný (zarotovaný bod) a celkovým výsledkem pak jsou přepočítané souřadnice markeru. Přepočítané detekované markery jsou označeny červeným čtvercem.

3.9.8 Detekce oblasti s odpověďmi

Tato část aplikace spočívá ve vymezení oblasti, v které se budou vyhledávat boxy. Slouží k tomu funkce *GetRoi(Marker *markers, Settings s)*. Jako vstup požaduje detekované markery a nastavení aplikace. Na základě údajů z konfiguračního souboru (x-ový a y-ový offsetu) se vymezí oblast. Výstupem je obdélník, definující tuto oblast.

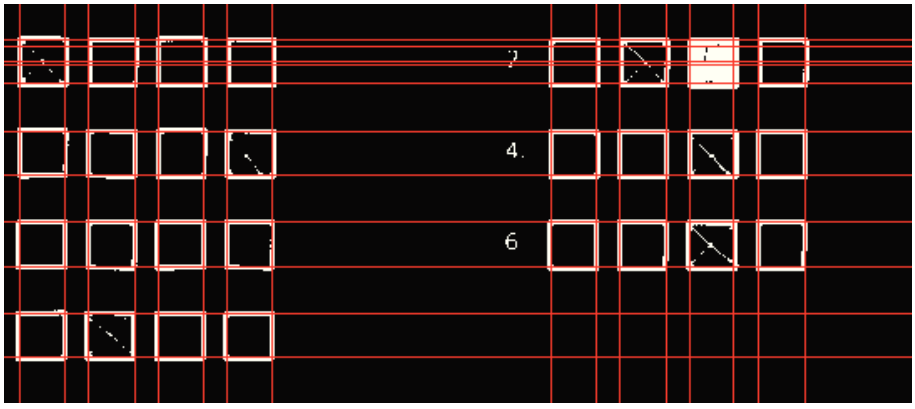
3.9.9 Detekce a označení odpovědí

Nyní přichází na řadu samotná detekce odpovědí. Pro tuto potřebu byla vytvořena funkce *DetectAnswers(IplImage* src, CvRect roi, Settings s)*. Vstupní parametry jsou: obrázek - test, oblast s odpověďmi a nastavení detektoru. Výstupem je datová struktura Result - obsah této struktury je popsán v kapitole 3.9.1.

Samotná detekce pracuje pouze s oblastí, definovanou parametrem funkce. Nejprve se provede předzpracování - vytvoří se invertovaný binární obraz. Na tento výstup se poté aplikuje Houghova transformace, pro vyhledání přímek. Výsledkem je množina úseček, reprezentována hodnotami $theta$ a r. Z těchto informací se vytvoří úsečka, která je reprezentována dvěma body a svou směrnicí. Dle směrnice se jednotlivé přímky rozdělí na vertikální a horizontální. Přímky, které nepatří ani do jedné kategorie, jsou zahozeny. Při detekci přímek hrozí problém duplicit - vlivem šumu, případně přetažením křížku přes hranice boxu. Proto je třeba nadbytečné úsečky odebrat. K tomu slouží funkce *RemoveDuplicateLines(Line *lines, int count, int vLines, int hLines, Settings s)*. Vstupem je pole úseček, jejich počet, informace o tom, zda se jedná o vertikální nebo horizontální úsečky a nastavení aplikace. Před samotným odstraněním přímek je aplikováno setřízení (od nejmenší x-ové, respektive y-ové souřadnice po největší).

Samotný proces vybrání neduplicitních přímek spočívá v porovnání nalezených přímek s referenčními. Postup je následující: jako první přímka se vždy zvolí první detekovaná přímka (tedy první přímka ze zadaného pole). Tato přímka je zároveň i první referenční a všechny ostatní referenční jsou pak vztaženy vůči této přímce. Jelikož jsou přímky setřízené, tak dostaneme hranici první řady (horní hrana), respektive prvního sloupce (levá hrana) boxů. Dalším krokem je posun na další řadu, respektive sloupec boxů a vytvoření referenční přímky. Posun je o přesně definovanou vzdálenost (definovanou v nastavení aplikace). Pokud pracujeme s horizontálními přímkami, tak je posun vždy

konstantní. U vertikálních přímek je třeba dávat pozor, na posun mezi otázkami - posun z posledního boxu první otázky na první box druhé otázky. Po vypočtení referenční přímky se porovnávají všechny přímky ze zadaného pole s touto referenční a vybírá se přímka, která má nejmenší vzdálenost od referenční. Tato přímka se přidá do výsledného seznamu přímek, neobsahující duplicity. Zároveň ale přímka musí splňovat kritérium, že vzdálenost od referenční přímky musí být menší než definovaná mez. Pokud toto kritérium není splněno, tak v případě horizontálních přímek to znamená konec - nebyly nalezeny žádné boxy. V případě vertikálních přímek se jedná o chybu detekce přímek a do výsledného pole se přidá referenční přímka. U vertikálních přímek algoritmus končí v případě, že našel dostatečný počet přímek (předpokládá se, že na každém řádku jsou 2 otázky a tudíž výsledná hodnota počtu přímek se vypočte jako $2\times$ počet možností odpovědí na jednu otázku). Výstupem funkce je struktura obsahující pole přímek bez duplicit a jejich počet.



Obrázek 10: Nalezené přímky před odstraněním duplicit a nežádoucích přímek

Nyní, když jsou odstraněny duplicity, se vypočítají všechny průsečíky vertikálních a horizontálních přímek. K tomuto vypočtu slouží funkce *ComputeIntersections(Line *hLines, int cHLines, Line *vLines, int cVLines)* která vrací pole typu CvPoint - pole průsečíků. Každý průsečík označuje levý horní bod boxu. Rozměr boxu je znám (definován v nastevní aplikace) a jsou tedy známy souřadnice jednotlivých boxů. Postupně se projdou všechny boxy a pomocí funkce *BoxEvaluation(IplImage* src, CvPoint p, Settings s)* se zjišťuje počet bílých pixelů. Vstupem je invertovaná binární verze zadaného obrázku, souřadnice levého horního bodu boxu a nastavení aplikace. Počet bílých pixelů boxu se vypočítává na základě histogramu. Výstupem je struktura, která obsahuje informace o daném boxu. Bližší informace o této struktuře lze nalézt v kapitole 3.9.1. Každý nalezený box je přidán do výsledné struktury.

Vždy po určitém počtu boxů (počet je dán nastavením aplikace - počtem možností odpovědi na otázku) přichází na řadu vyhodnocení otázky. Jelikož při výpočtu průsečíku se nezohledňuje fakt, že může být počet otázek lichý (tedy že poslední řada má jen jednu otázku a ne dvě), tak může vzniknout problém, že v místě průsečíku nebude žádný box. Aby se tomuto problému předešlo, tak se ukládá počet bílých pixelů boxu a při vyhodno-

Obrázek 11: Nalezené boxy (zelené přímky označují referenční přímky)

cení otázky se sečtou všechny pixely všech boxů, patřících k dané otázce. Pokud je počet nižší, než stanovený limit, pak se jedná o prázdný prostor a výsledek není započítán.

Rozhodnutí o odpovědi na otázku probíhá na základě vzájemného porovnání sekvence boxů. Postup je následující: nejprve se všechny boxy z dané sekvence vzestupně setřídí. Pak proběhne odstranění začerněných boxů - vezme se box na nejvyšší pozici a vypočítá se procentuální zastoupení bílých pixelů v boxu a pokud je hodnota vyšší než určitý limit, tak je box označen jako začerněný. Postup se opakuje, dokud se nenajde box jehož zastoupení pixelů je nižší nebo je k dispozici jen jeden prvek. V případě, že odstranění těchto boxů zůstane jen jeden prvek, tak se opět provede porovnání s limitem. Pokud je nad, tak je otázka prohlášena za nezodpovězenou, v opačném případě se box bere jako zaškrtnutý. Po odstranění nežádoucích boxů v sekvenci zůstanou jen zaškrtnuté a nezaškrtnuté boxy. Jelikož je sekvence setřízená a odpověď na otázku může být vždy jen jedna, tak se porovnává box na poslední pozici (respektive počet pixelů boxu) s předposledním boxem v sekvenci. Pokud je hodnota větší než určený limit, pak je box na poslední pozici označen jako zakřížkovaný - do výsledné struktury se uloží číslo otázky, odpověď a souřadnice boxu. V případě, že limit bude menší, pak nastává případ, že buď nebylo odpovězeno na otázku nebo jsou zakřížkované minimálně 2 odpovědi - otázka se prohlásí za nezodpovězenou.

3.9.10 Uložení výsledků

V předchozí části byly zjištěny všechny potřebné informace (otázky na jednotlivé odpovědi, souřadnice boxů s odpověďmi a souřadnice všech boxů) a nyní nastává poslední krok - uložení těchto informací do souboru, který bude sloužit jako zdroj dat při přidávání testů do databáze webové aplikace. Aby nevznikaly konflikty při přidávání

LOG001	login studenta
7S0VD	verze zadání
SPJA	předmět
2011-12-12	datum
4	počet možností odpovědi na jednu otázku
24	rozměr boxu
ABDCBCB	odpovědi na otázky
241	x-ová souřadnice levého horního bodu prvního boxu
419	y-ová souřadnice levého horního bodu prvního boxu
282	x-ová souřadnice levého horního bodu druhého boxu
419	y-ová souřadnice levého horního bodu druhého boxu
...	souřadnice zbylých boxů

Tabulka 1: Příklad výstupního souboru

testů do databáze, tak má každý soubor unikátní jméno - tvořeno identifikačními údaji. Název souboru (jak textového souboru s daty tak obrázku) je ve tvaru: login;verze;datum; predmet.

Uložení je realizováno funkcí *SaveResults(char *filename, Settings s, Result r, Login login, Version version, Subject subject, Date date)*. Parametry jsou: jméno výstupního souboru, nastavení aplikace, struktura obsahující výsledky detektoru a údaje z QR kódu. Strukturu výstupního souboru zobrazuje následující tabulka (levý sloupec reprezentuje řádek souboru a pravý komentář) 1:

4 Webové rozhraní

Výstupem aplikace je textový soubor obsahující data z detekce a obrázek s označenými odpověďmi. Nyní je potřeba tyto informace vhodně prezentovat a zpřístupnit výsledky pedagogům a studentům. Ze specifikace zadání (má se jednat o webové rozhraní) a předběžné analýzy výstupních dat, byl pro implementaci vybrán Django framework. Integrace se službami VŠB (přihlášení LDAP) je realizována knihovnou Python-LDAP.

4.1 Django framework

Django je webový framework napsaný v jazyce Python. Začal vznikat v roce 2003 a původně se jednalo o součást redakčního systému kansaského deníku Lawrence Journal-World. V roce 2005 byl tento framework vydán jako open source, pod licencí BSD.

Název vznikl odvozením od uměleckého jména jazzového kytaristy Django Reinhardta, který byl označován za průkopníka žánru.

Účelem Djanga je, aby se co nejvíce věcí automatizovalo.

Hlavní principy frameworku:

- DRY (Don´t Repeat Youself) - Princip, který zabraňuje rutinnímu opakování kódu. Využívá se k tomu objektově orientované programování a automatické generování kódu.
- MTV (Model-Template-View) - Přístup, odvozen z architektury MVC. Jedná se o oddělení definice dat, zpracování a jejich prezentace. Základem je databázový model, z kterého získáváme data (view), které se posílají HTML šabloně.
- ORM (Object-Relational Mapping) - Django používá objektový přístup k databázi. Nepoužívají se SQL dotazy, ale místo nich pracujeme pouze s objekty a funkcemi Pythonu. Možnost psát SQL dotazy je zde ponechána.

Některé další rysy Djanga: administrační rozhraní je generováno automaticky, šablonovací systém, generování formulářů z databázového modelu, automatická validace, podpora kešování a odlehčený webový server.

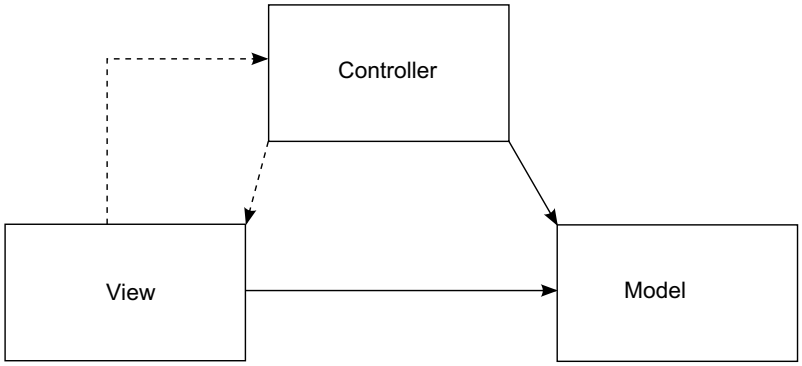
4.2 Model-View-Controller

Architektura MVC se dělí na 3 základní logické celky: model, view a controller.

Cílem je, aby každá z těchto částí byla nezávislá na ostatních - aby dopad změn na ostatní části, při úpravě jedné části byl co nejmenší.

Model specifikuje data s nimiž aplikace pracuje. View (pohled) zobrazuje uživatelské rozhraní, tedy převádí data, která jsou reprezentována modelem, do podoby vhodné pro prezentaci. Controller (řadič) má na starosti aplikační logiku - zajišťuje změny v modelu nebo pohledu. Typicky se může jednat o reakce na události přicházející od uživatele.

V navržené webové aplikaci představují modely jednotlivé databázové modely. View je reprezentován HTML šablonou a jako controller slouží Python funkce, starající se o zpracování dat z databáze a jejich předání HTML šabloně.



Obrázek 12: Architektura MVC [24]

Na obrázku (12) je ukázáno propojení jednotlivých částí:
Z obrázku (12) lze vyčíst, že existují dvě přímé vazby:

- Controller - Model. Vazba umožňující controlleru úpravu dat.
- View - Model. Vazba sloužící pro zobrazení dat.

V praxi (a v závislosti na konkrétní variaci MVC) může ještě existovat vazba controllerem a view a to buď jednosměrná nebo obousměrná.

Nikdy ale nesmí existovat přímá vazba mezi modelem a ostatními komponentami. Může se využít maximálně nepřímá vazba, kdy při změně dat modelu se příslušnému view pošle upozornění (např. pomocí návrhového vzoru observer).

Architektura MVC dnes nachází nejčastější uplatnění ve webových technologiích.

Existuje mnoho variací MVC, které mohou být realizovány odlišným způsobem [24]. Obecný princip lze popsat následovně:

1. Akce provedená uživatelem (např. kliknutí na odkaz nebo zmáčknutí tlačítka).
2. Controller dostane oznámení o provedené akci.
3. Controller v případě potřeby aktualizuje model (např. aktualizace seznamu nakoupeného zboží).
4. Zpracování aktualizovaných dat (např. přepočítání celkové ceny).
5. View zobrazí na základě aktualizovaného modelu data.
6. Čeká se na další akci uživatele a cyklus se opakuje.

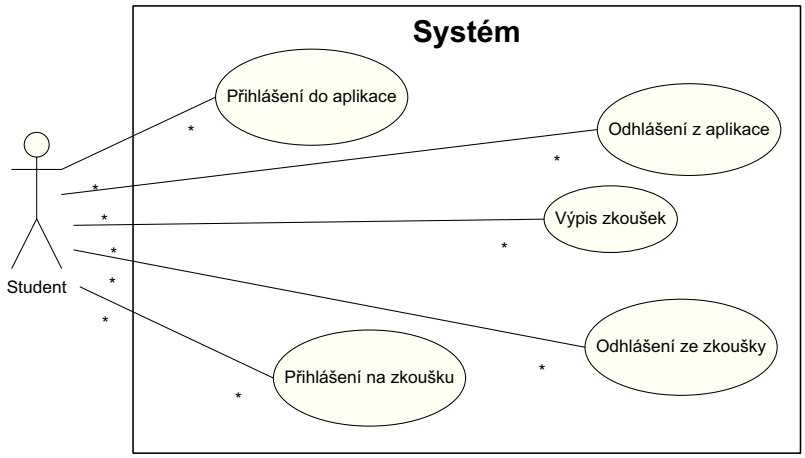
4.3 Diagram případů užití (Use-Case Diagram)

Účelem Use-Case diagramu je specifikovat vztah mezi uživateli systému a funkcemi, které systém nabízí - definující funkcionalitu systému [25]. Diagram je tvořen:

- Aktéry - definují uživatele nebo jiné systémy, které budou komunikovat se softwarem.
- Případy užití - specifikují jednotlivé funkce, které můžou aktéři provádět.

Mezi případy užití mohou existovat následující relace:

- Uses - označení definující, že nějaký případ užití je využíván i jinými případy užití.
- Extends - definuje, že jeden případ užití je rozšířen jiným případem užití.
- Generalizace - definuje vztah mezi obecným případem užití a konkrétním odvozeným případem. Generalize nemusí být jen mezi případy užití, ale může také existovat mezi aktéry.



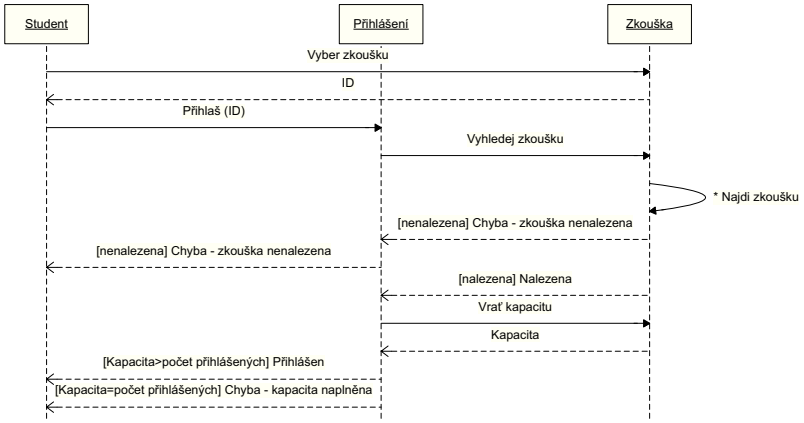
Obrázek 13: Příklad Use-Case diagramu

4.4 Sekvenční diagram

Sekvenční diagram popisuje interakci mezi objekty - zprávy, které si objekty posílají v čase. Diagram je tvořen objekty (uspořádanými do sloupců) a zprávami, které si posílají (znázorněnými šipkami). Zpráva může být buď synchronní - odesílatel čeká na odpověď a až poté pokračuje v provádění činnosti, nebo asynchronní - odesílatel odešle zprávu a ihned pokračuje ve své činnosti. Odpověď adresáta je modelována tzv. návratovou zprávou, znázorněnou přerušovanou čárou. Časová osa plyne shoda dolů [25].

4.5 Požadavky na systém

Hlavním požadavkem bylo vytvořit přehledný webový systém, který by se staral o správu a prezentaci výsledku z detektoru, který byl navržen v kapitole 3 a který bude integrován se službami VŠB (přihlášení LDAP).



Obrázek 14: Příklad sekvenčního diagramu

Při analýze požadavků se bral především ohled na uživatelské role, které budou se systémem pracovat. Jedná se hlavně o role student a učitel.

Pro studenta by měl systém sloužit jako přehled vyplněných testů, zpracovaných detektorem. Mezi další požadavky patřilo vytvoření interaktivního prostředí pro nahlášení špatné detekce odpovědi v testu.

Učitel by měl mít možnost generovat testy, přidávat do databáze nové testy a reagovat na chybné detekce, nahlášené studenty.

Bližším popisem práv jednotlivých typů uživatelů se zabývá následující kapitola.

4.6 Uživatelské role

V systému jsou definovány 4 základní uživatelské role: administrátor, garant, učitel a student.

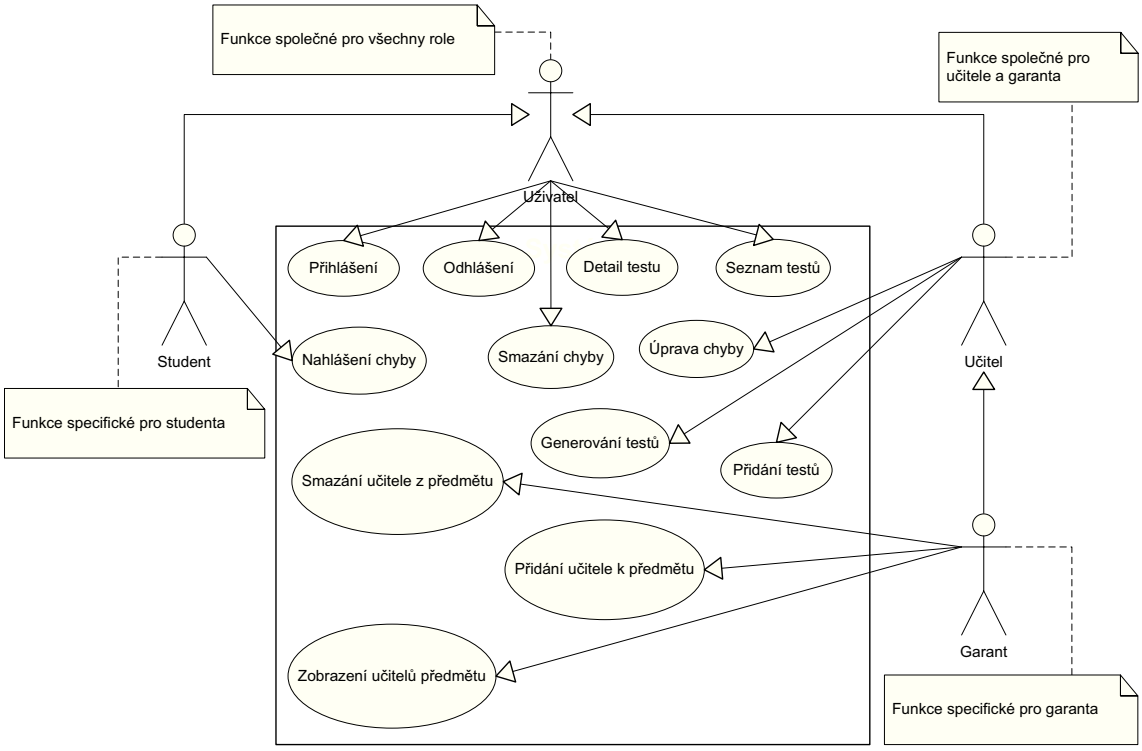
Administrátor má největší pravomoce, jako jediný může přidávat garanty (respektive povýšit učitele na garanta). V případě potřeby může i mazat a upravovat garanty jednotlivých předmětů. Dále má za úkol celkovou správu aplikace. Může v případě potřeby upravovat data jednotlivých testů, přidávat a mazat předměty a spravovat případné chyby vzniklé při zpracování výsledků z detektoru. Jako jediný může do databáze přidávat nové předměty. Administrátor může být pouze jeden.

Garant má stejné práva jako učitel - tedy prohlížení testů, které přidal, úprava a mazání nahlášených chyb u jednotlivých testů, generování a přidávání nových testů. Oproti učiteli má možnost zobrazit všechny testy z předmětů, které garantuje. Další pravomocí je, že může k předmětům, které garantuje, přidávat nové učitele nebo je odebírat. Garant může být pro každý předmět vždy jen jeden.

Učitel si může prohlížet testy dle jednotlivých předmětů, ale je omezen tím, že uvidí jen testy, které sám přidal. Také je zde ponechána možnost vypsat si všechny testy, které přidal, bez ohledu na předmět. U každého testu lze zobrazit detail - výsledek z detektoru a seznam chyb, nahlášených studenty. Učitel má práva upravovat a mazat nahlášené chyby. Dále může učitel generovat testy a přidávat do databáze již vyplněné testy.

Student si může prohlížet testy, které vyplnil. Pro větší přehlednost jsou má možnost vybrat si jen testy z určitého předmětu, ale zároveň je zde ponechána možnost zobrazit všechny testy, bez ohledu na předmět. U každého testu je možno zobrazit detail - výstup z detektoru, s označenými odpověďmi a v případě špatné detekce odpovědi má možnost pomocí vybrání příslušného boxu v obrázku testu nahlásit chybu. U každého testu jsou zobrazeny detekční chyby, které student nahlásil.

Následující use case diagram názorně zobrazuje práva uživatelských rolí (15):



Obrázek 15: Use Case diagram

4.7 Modely

Model v Djangu představuje popis, jak má databáze vypadat uvnitř. Z modelu se vytváří schéma databáze - třída odpovídá názvu tabulky a atributy odpovídají jednotlivým sloupcům. Model je reprezentován pomocí Pythonovských tříd a atributů. Vazby a datové typy se realizují speciálními třídami.

V aplikaci jsou definované následující modely (primární klíč je ve všech modelech vytvořen automaticky a je označen id):

- Garant - pomocná tabulka, obsahující garanty jednotlivých předmětů. Protože garant pro předmět může být vždy jen jeden, tak předmět v tabulce musí být unikátní.

```
class Garant(models.Model):
```

```
    login = models.CharField(max_length=10)
    subject = models.ForeignKey(Subject, unique=True)
```

Výpis 1: Model Garant

- Subject - tabulka obsahující všechny předměty. Má atributy: název předmětu, jeho kód a zkratka. Všechny atributy musí být unikátní.

```
class Subject(models.Model):
    name = models.CharField(max_length=50, unique=True)
    code = models.CharField(max_length=50, unique=True)
    shortcut = models.CharField(max_length=10, unique=True)
```

Výpis 2: Model Predmet

- Teacher_Subject - pomocná tabulka, zachycující, který učitel učí jaký předmět. Omezením je, že tato dvojice musí být unikátní.

```
class Teacher_Subject(models.Model):
    login = models.CharField(max_length=10)
    subject = models.ForeignKey(Subject)

    class Meta:
        unique_together = ('login', 'subject')
```

Výpis 3: Model Ucitel_Predmet

- Test - uložení testů, které prošly detektorem. Každý test je asociován právě s jedním datovým souborem a jedním obrázkem a proto musí být jak soubor, tak obrázek v tabulce unikátní. Každému testu je přiřazeno bodování, dle kterého se vypočítávají získané body.

```
class Test(models.Model):
    student = models.CharField(max_length=10)
    subject = models.ForeignKey(Subject)
    version = models.CharField(max_length=10)
    date = models.DateField()
    answers = models.CharField(max_length=50)
    teacher = models.CharField(max_length=10)
    answer_count = models.IntegerField()
    scoring = models.ForeignKey(Scoring)
    score = models.IntegerField()
    file_name = models.CharField(max_length=50, unique=True)
    image = models.CharField(max_length=50, unique=True)

    class Meta:
        unique_together = ('subject', 'date', 'version')
```

Výpis 4: Model Test

- Error - tabulka obsahující chyby v detekci, nahlášené studenty. Dvojice test a jeho chyba musí být unikátní.

```
class Error(models.Model):
    test = models.ForeignKey(Test)
    question = models.IntegerField()
    answer = models.CharField(max_length=1)
    new_answer = models.CharField(max_length=1)

    class Meta:
        unique_together = ('test', 'question')
```

Výpis 5: Model Chyba

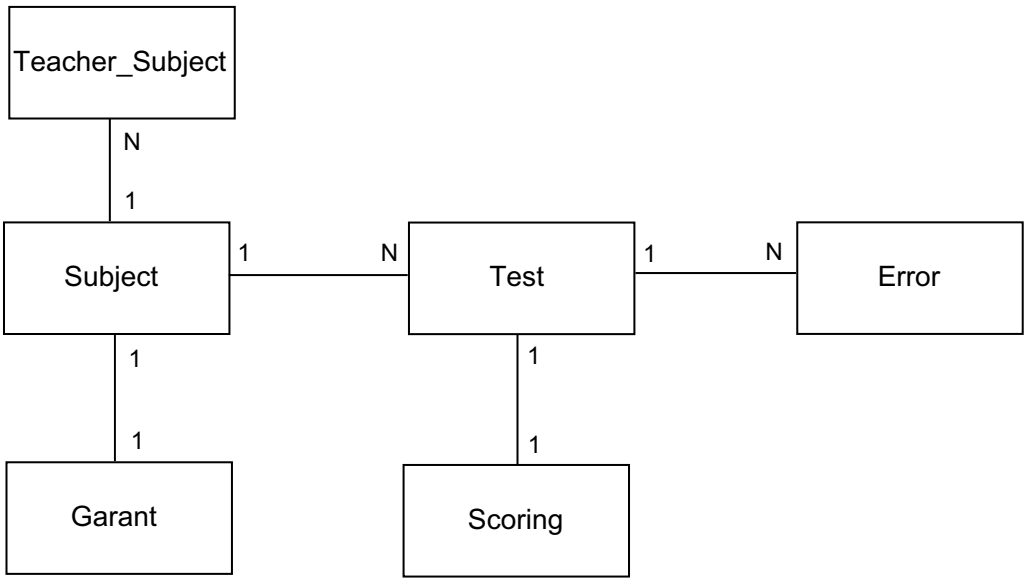
- Scoring - tabulka obsahující správné odpovědi a bodové ohodnocení pro každou vygenerovanou verzi testu.

```
class Scoring(models.Model):
    subject = models.ForeignKey(Subject)
    date = models.DateField()
    version = models.CharField(max_length=10)
    right_answers = models.CharField(max_length=50)
    points = models.CharField(max_length=50)

    class Meta:
        unique_together = ('subject', 'date', 'version')
```

Výpis 6: Model Bodovani

Na obrázku (16) je znázorněn databázový model, který ukazuje vzájemné propojení modelů:



Obrázek 16: Databázový model

Z obrázku (16) a z modelů lze vyčíst následující vazby:

- Každý předmět může být garantován pouze jedním učitelem, ale jeden učitel může být garantem několika předmětů.
- Učitel může učit více předmětů.
- Každý test je z jednoho předmětu a může ho zadat jen jeden učitel.
- Chyba je asociována s jedním testem, ale každý test může obsahovat několik chyb.
- Test je vyplněn právě jedním studentem, ale daný student může vyplnit více testů.

4.8 Implementace webu

Pro všechny moduly jsou společné dvě funkce - přihlášení do aplikace a odhlášení.

Jak již bylo zmíněno výše, je přihlášení integrováno se službami VŠB - autorizace uživatele probíhá vůči LDAPu. Po zadání přihlašovacích údajů se nejprve vytvoří anonymní spojení s LDAP serverem pro účely vyhledání informací, které jsou potřeba pro autorizaci. Na základě zadaného loginu, hesla a vyhledaných informací se vytvoří spojení - provede se autorizace. Pokud je úspěšná, tak funkce vrací informaci zda se jedná o studenta nebo učitele a jeho jméno. V případě neúspěchu se vrací zpráva o neúspěšném přihlášení. Po úspěšném přihlášení se do cookies uloží informace o loginu a jméně uživatele a proběhne přesměrování na hlavní menu studenta, případně učitele.

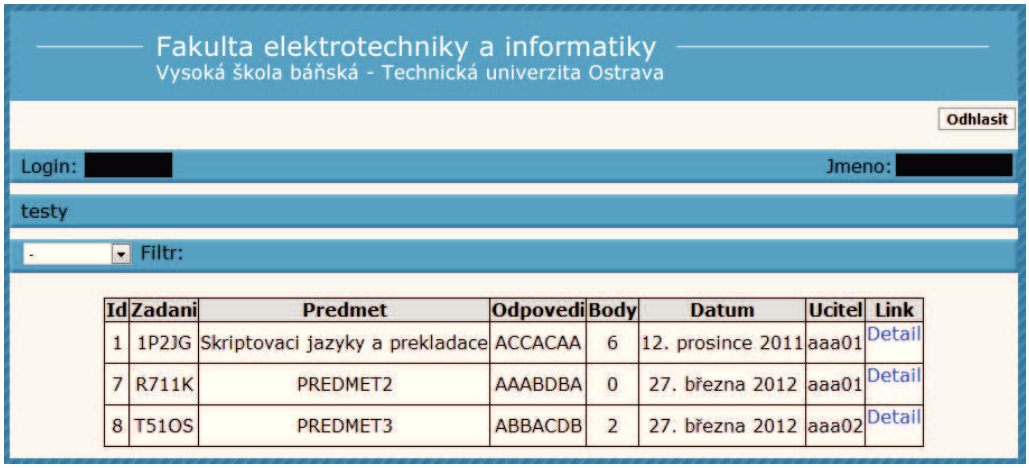
Při odhlašování z aplikace se mažou veškeré cookies informace, vytvořené v průběhu prohlížení webu a u učitele se také mažou vytvořené dočasné adresáře.

4.8.1 Modul Student

V hlavním menu studenta je zobrazena tabulka, obsahující všechny testy, které daný student absolvoval. Obsahují informaci o předmětu, datumu, verzi zadání, počtu bodů, jednotlivých odpovědí a loginu učitele, který přidal test do databáze. Je zde i možnost zobrazit si detail testu.

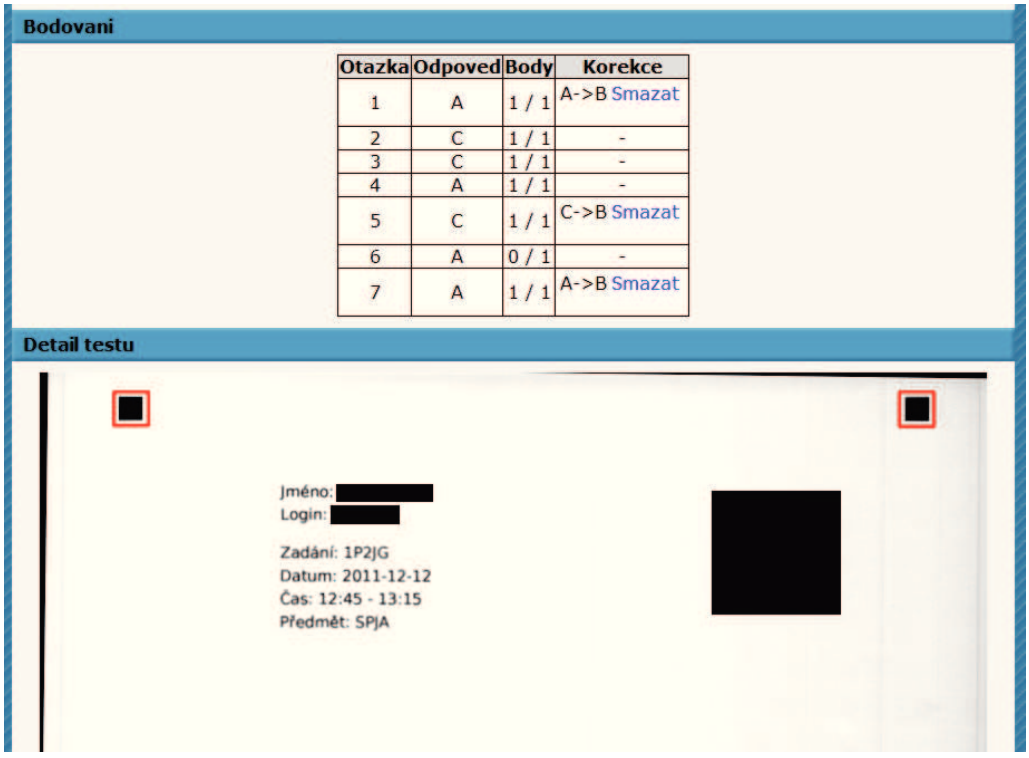
Student má možnost filtrovat si testy dle předmětu. Vedle výběru předmětu, dle kterého se bude filtrovat, je uveden aktuálně používaný filtr.

V detailu testu je blíže rozepsáno bodování testu - v tabulce je zobrazeno číslo otázky, odpověď studenta, počet získaných bodů/počet bodů za správnou odpověď a korekce. Pod touto tabulkou je zobrazen výstup z detektoru - student má možnost vidět označené odpovědi a v případě špatné detekce kliknutím na box otázky nahlásí chybu. Při nahlášení chyby se odešle informace o testu a čísle boxu, na který student kliknul. Z čísla boxu se podle počtu možných odpovědí na otázku zjistí číslo otázky, aktuální odpověď a hodnota nové odpovědi. Pokud pro daný test a otázku již není nahlášená nějaká chyba, tak se chyba uloží do databáze a v tabulce s bodováním se do sloupce korekce přidá dvojice aktuální odpověď-nová odpověď, signalizující nahlášenou chybu. Pokud by se student překliknul nebo by špatně nahlásil chybu, tak má možnost smazat nahlášenou chybu. Informace o pozicích boxů nejsou uloženy v databázi, ale vždy při zobrazení detailu se načítají z příslušného textového souboru, obsahujícího data o testu (výstup z detektoru).



Id	Zadani	Predmet	Odpovedi	Body	Datum	Ucitel	Link
1	1P2JG	Skriptovaci jazyky a prekladace	ACCACAA	6	12. prosince 2011	aaa01	Detail
7	R711K	PREDMET2	AAABDBA	0	27. března 2012	aaa01	Detail
8	T51OS	PREDMET3	ABBACDB	2	27. března 2012	aaa02	Detail

Obrázek 17: Student - zobrazení testů



Otazka	Odpoved	Body	Korekce
1	A	1 / 1	A->B Smazat
2	C	1 / 1	-
3	C	1 / 1	-
4	A	1 / 1	-
5	C	1 / 1	C->B Smazat
6	A	0 / 1	-
7	A	1 / 1	A->B Smazat

Obrázek 18: Student - zobrazení detailu testu

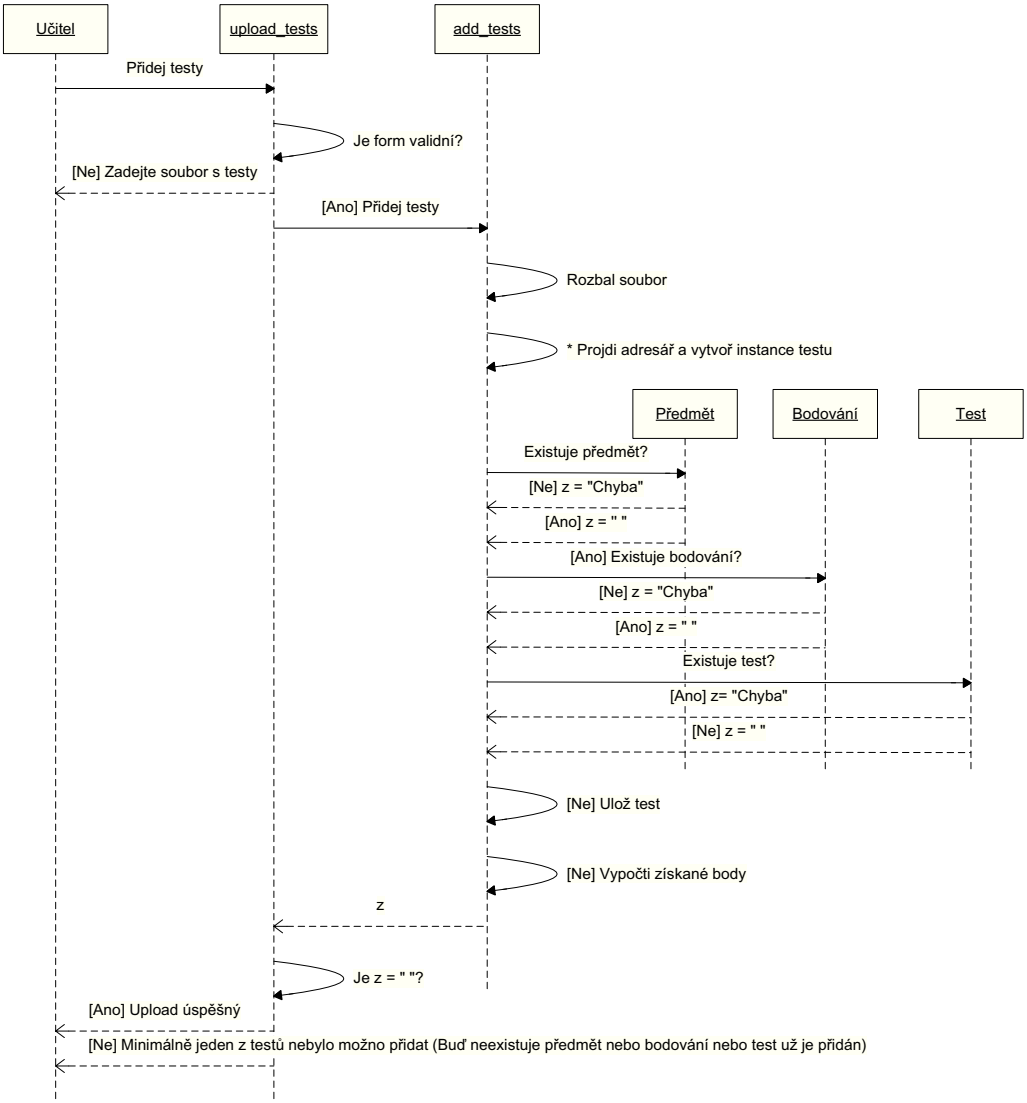
4.8.2 Modul Učitel

Po přihlášení se zobrazí hlavní menu, kde má možnosti - přidat nové testy, generovat testy a zobrazit přidané testy.
Pro přidání nových testů slouží formulář, kterým si uživatel vybere soubor, obsahující

testy, které se mají uploadovat. Aby se přidání testů provedlo úspěšně, tak vybraný soubor musí být zip a pojmenování zipu a adresáře uvnitř musí být shodné. Adresář musí obsahovat vždy textový soubor obsahující informace o testu a příslušný obrázek, který je pojmenován stejně jako textový soubor, který bude zobrazen v detailu testu. Aby nevznikaly konflikty, kdyby více uživatelů najednou přidávalo testy a náhodou by byly shodné jména zipů - tudíž by mohlo dojít k přepsání dat, tak byl tento problém vyřešen vytvořením pomocných adresářů. V adresářích temp a data se vytvoří dočasné adresáře, pojmenované loginem uživatele. Zip soubor se nejprve uploaduje do složky pracovního adresáře v tempu a pak se rozbalí do uživatelského adresáře v datech. Postup při přidávání testů do databáze je následující: projde se rozbalený zip, pokud se narazí na textový soubor, tak se přečte jeho obsah, vytvoří se instance testu a ta se uloží do databáze. Zároveň se zkopíruje textový soubor, respektive obrázek do určených adresářů. Pokud už test v databázi existuje, tak se vypíše chybové hlášení. V opačném případě se vypíše, že přidání bylo úspěšné. Přidávat lze pouze testy z předmětů, které jsou v databázi. Při úspěšném přidání testu se provede výpočet získaných bodů. Každému testu se podle jednoznačných údajů přiřadí bodování (správné odpovědi a počty bodů), dle kterého se provede výpočet získaných bodů. Po nahrání testů se dočasné adresáře smažou.

Po kliknutí na položku generovat testy, se objeví formulář pro zadání informací pro vygenerování testů. Předmět lze vybrat jen z předmětů, které daný učitel učí. Čas testu a čas určený na vypracování se zadávají jako řetězce. Při zadávání datumu se po kliknutí na příslušnou kolonku zobrazí kalendář pro výběr datumu. Počet variant značí, kolik verzí testů se má vytvořit. Počet odpovědí označuje kolik možností odpovědí na jednu otázku bude. U počtu otázek je možné otázky vybírat náhodně, z jedné množiny otázek nebo vytvořit několik kategorií a z každé kategorie vybrat několik otázek. V prvním případě, kdy je pouze jedna kategorie, stačí zadat jen počet otázek. U druhého případu je třeba počet otázek z jednotlivých kategorií oddělit středníkem. Například budou 3 kategorie a z každé budeme chtít 2 otázky. Počet otázek by se zadal jako řetězec: ”2;2;2”. Poslední údaje, které je třeba zadat při generování, jsou soubor se studenty a soubor s otázkami. Soubor se studenty musí být textový (csv soubor) s přesně definovanou syntaxí: ”číslo;login;jméno”. Jeden student na jednom řádku. Otázky musí být zadány ve formě textového souboru, který má také předepsanou syntaxi. Otázka musí být ve tvaru: ”Otázka#odpověď 1#odpověď 2#odpověď 3#odpověď 4#počet bodů#správná odpověď”. Počet odpovědí musí souhlasit s počtem zadaným při generování, jinak aplikace oznámí chybu. Počet bodů je reprezentován celým číslem a správná odpověď znakem ”A”,”B”atd. Je možno zadávat otázky, které jsou přes více řádků. Otázky jsou odděleny znakem ”#*”, zapsaným na nový řádek. Další otázka pak začíná zase na novém řádku. Kategorie se oddělují znakem ”##”, zapsaným nový řádek. Při generování se kontroluje počet odpovědí, zjištěných ze zadaného souboru a porovnává se ze zadanou hodnotou a také se počítá počet otázek v kategorii a v případě, že uživatel chce víc otázek z kategorie, než je dostupných, tak se vypíše chyba. Následující sekvenční diagram blíže popisuje postup při generování testů (20:):

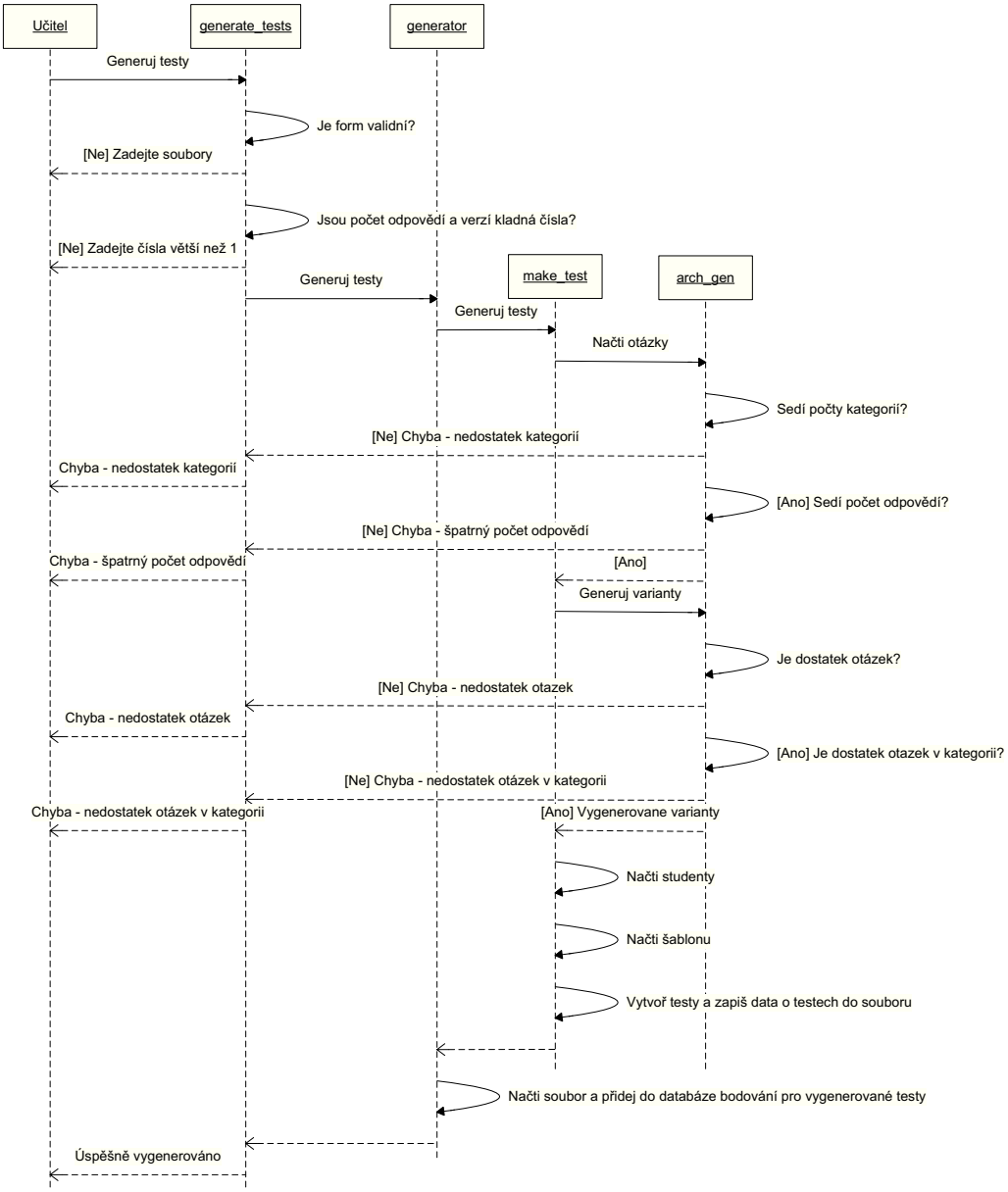
Víceuživatelský přístup je opět vyřešen vytvořením dočasných adresářů, které se po vygenerování smažou (kromě adresáře obsahující výstup generátoru). Výstupem ge-



Obrázek 19: Sekvenční diagram zjednodušeně zobrazující průběh přidání testů

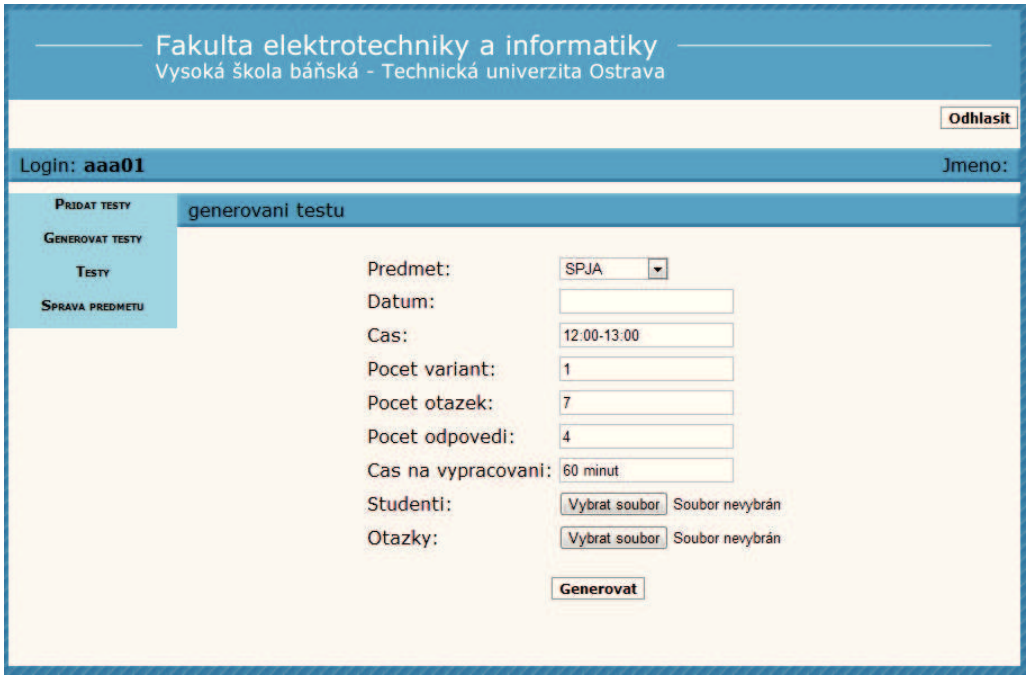
nerátoru je zip soubor, obsahující pdf soubory se zadáním testů a arch pro označení odpovědí. Při úspěšném vygenerování je vypsáno oznámení o úspěšném vygenerování a vytvořen link pro stažení zipu. Vedlejším výstupem je textový soubor s informacemi o správných odpovědích a bodech. Pro každý vygenerovaný test se do databáze přidají informace o bodování, které slouží pro výpočet získaných bodů při nahrání testu do databáze.

Položka testy slouží k zobrazení testů, které učitel přidal. Nejprve se objeví seznam vyučovaných předmětů, který slouží jako filtr testů dle předmětu. Po kliknutí na předmět se zobrazí všechny nahrané testy z tohoto předmětu. Objeví se tabulka, obsahující základní informace o testu - verze zadání, datum, čas, počet získaných bodů, login studenta, který

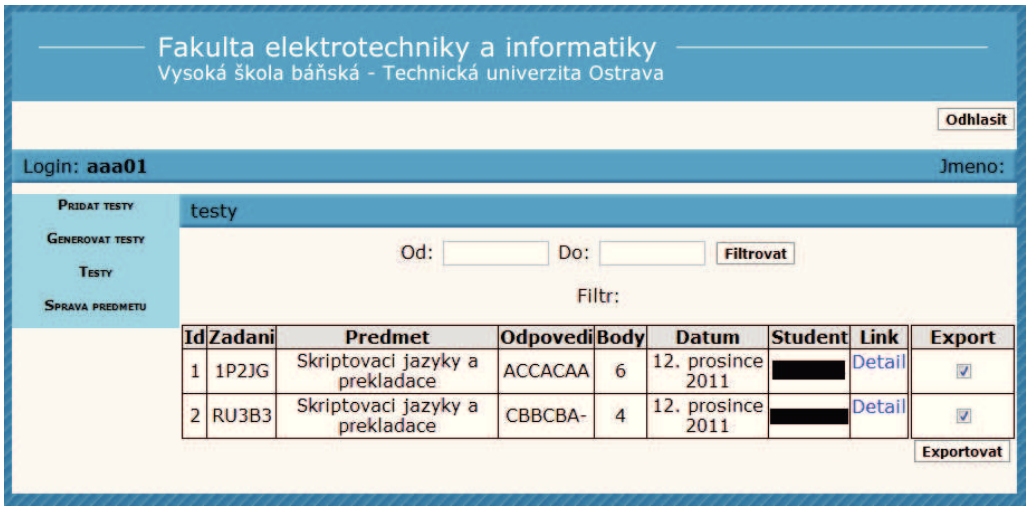


Obrázek 20: Sekveční diagram zjednodušeně zobrazující průběh generování testů

test vyplnil a odkaz na detail testu. Filtrovat lze dle datumu - zadáním od kdy do kdy. Při zobrazení testů jen z určitého předmětu ma učitel možnost exportovat vybrané testy. Výsledkem exportu je textový (csv) soubor, pojmenovaný dle předmětu, z kterého jsou testy a obsahující login studenta a jeho body na každém řádku. Data jsou ve tvaru ”login;body”. Po úspěšném exportu se zobrazí link pro stažení souboru. Učitel má také možnost zobrazit si všechny testy, které zadal.



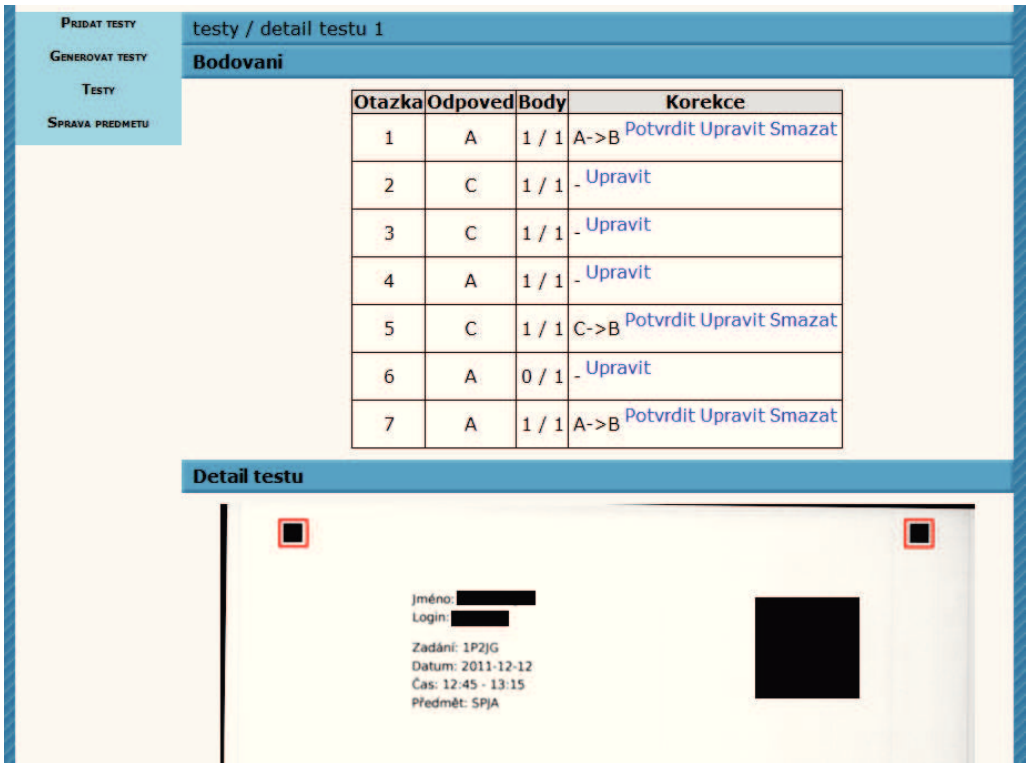
Obrázek 21: Učitel - formulář pro generování testů



Obrázek 22: Učitel - zobrazení testů

Kliknutím na detail testu se zobrazí okno s detailem daného testu. Stejně jako u studenta, je zde zobrazena tabulka, podrobněji zobrazující získané body - číslo otázky, odpověď studenta, získané body / počet bodů za správnou odpověď a korekce. Sloupec korekce obsahuje chybné detekce, nahlášené studentem. Učitel má možnost chybu potvrdit - v tomto případě se upraví data testu, přepočítají se body a chyba se smaže. Dále

chybu smazat nebo ji upravit. Při úpravě má možnost libovolně upravit odpověď. Opět se upraví data testu, přepočítají se body a chyba se smaže. Zároveň má učitel možnost upravit odpověď na jakoukoliv otázku z testu (tedy i otázku, u které student nenahlásil chybu detekce). Pod tabulkou zobrazující body je zobrazen náhled testu s detekovanými odpověďmi.



Obrázek 23: Učitel - zobrazení detailu testu

4.8.3 Modul Garant

Garant má společné funkce s učitelem, ale má jednu navíc - správu garantovaných předmětů. Zda je učitel garantem nebo ne, se zjišťuje z tabulky Garant. Pokud je nalezen aspoň jeden záznam, s daným loginem učitele, tak se jedná o garanta.

Po kliknutí na položku správa předmětů se objeví seznam garantovaných předmětů. Kliknutím na předmět se objeví jednoduché menu, kde může garant přidávat nové učitele k předmětu nebo je odebírat.

5 Testování

Prvním krokem při vytváření detektoru byla detekce optických značek, definujících oblast s daty - tedy detekce markerů. První pokusem bylo vytvoření binárního obrazu a detekce pomocí horizontální a vertikální projekce pixelů. A určení pozice markerů jako průnik horizontální a vertikální projekce. Toto řešení přineslo neuspokojivé výsledky a proto se přešlo na druhou možnost - detekce rohů. První pokusem bylo umístění malého detekčního okna na pevnou pozici do rohu obrazu. Zde nastaly problémy s okrajem papíru - detektor zde nacházel rohy a výsledkem byly falešné detekce. Následovala úprava algoritmu - detekční okno nebylo umístěno napevno, ale pohybovalo se po omezené části obrazu, tak dlouho, dokud nebyly nalezeny právě 4 rohy nebo nebyl překročen definovaný maximální počet cyklů. Toto řešení již přineslo uspokojivé výsledky.

Tím byla definována oblast s daty a mohlo se přejít na samotnou detekci odpovědí. Nejprve byla definovaná oblast rozdělena na dvě části - oblast s informacemi identifikujícími test (login studenta, verze zadání, datum a předmět) a oblast se samotnými daty. A nad touto oblastí se provedla Houghova transformace. Aby byly výsledky co nejlepší, bylo třeba nastavit co nejideálnější level tresholdu. Při nízkém levelu se vyskytlo mnoho falešné detekce - i místa kde byly popisy otázek (čísla otázek a varianty odpovědí) detekoval čáry. Nastavením vyššího levelu se tento problém vyřešil, ale objevil se jiný. Transformace měla velké problémy detekovat vertikální čáry. Zkoušely se různé varianty nastavení tresh levelu u Cannyho detektoru i u samotné Houghovy transformace, ale žádná z nich nepřinesla uspokojivé výsledky. Přešlo se tedy na druhou možnost. Oblast s daty se ořezala, aby v ní byly jen boxy s odpovědí a ne popisy. U vyššího tresh levelu se objevil stejný problém jako předtím, takže se muselo přejít na nižší. Výsledky nebyly ideální, vzniklo velké množství multidetekce a částečně i falešné detekce. S multidetekcí si program dokáže poradit a to tedy nevadilo. Falešné detekce bylo podstatně méně než v předchozím případě a vznikly výsledky, s kterými se už dalo bez větších potíží pracovat. Alternativní možností bylo použit jako vstup Houghovy transformace binární obraz, místo Cannyho detektoru. Dosáhlo se ještě lepších výsledky a proto byla nakonec zvolena tato varianta.

Poté přišlo na řadu testování samotných boxů. První pokusem bylo aplikování Cannyho detektoru hran na vstupní obraz. Výstup Cannyho detektoru sloužil jako zdroj při evaluaci (zjištění zda je box zaškrtnutý nebo ne) boxů. Rozhodnutí o evaulaci probíhalo na základě porovnání počtu bílých pixelů v boxu s nastavenými limity. Výsledky shrnuje následující tabulka 2:

Počet boxů	280
Počet správných detekcí	271
Počet špatných detekcí	9

Tabulka 2: Výsledky evaluace boxů nad Cannyho detektorem

Z tabulky 2 lze vyčíst, že úspěšnost byla celkem vysoká, ale vzhledem k malému počtu testovacích dat (10 testů, každý měl 7 otázek a každá otázka 4 možné odpovědi),

nelze výsledky označit za přijatelné. Úprava tresh levelu pomohla odstranit některé špatné detekce, ale objevily se nové chyby.

Místo vstupu pro evaluaci se použil invertovaný binární obraz vstupního obrazu. Postup byl stejný - opět se vytvořil histogram boxu a počítal se počet bílých pixelů. Výsledky jsou shrnuty v následující tabulce:

Počet boxů	280
Počet správných detekcí	280
Počet špatných detekcí	0

Tabulka 3: Výsledky evaluace boxů nad invertovaným binárním obrazem

Dosáhlo se 100% úspěšnosti. Ale vzhledem k malému počtu testů, nelze výsledky považovat za objektivní.

Všechny výše uvedené výsledky byly na základě testů, naskenovaných jedním skenerem. To by však bylo velice omezující a proto se testovaly i testy, naskenované různými skenery. Zde systém evaluace selhal - u testů z prvního skeneru byla sice úspěšnost 100%, ale u testů z druhého se úspěšnost blížila 0%. Proto byl pozměněn princip zjišťování odpovědi na otázku z počítání pixelů a porovnání s limitem na vzájemné porovnání boxů. Dosažené výsledky shrnuje tabulka 4:

Počet otázek	106
Počet správně detekovaných odpovědí	103
Počet špatně detekovaných odpovědí	3

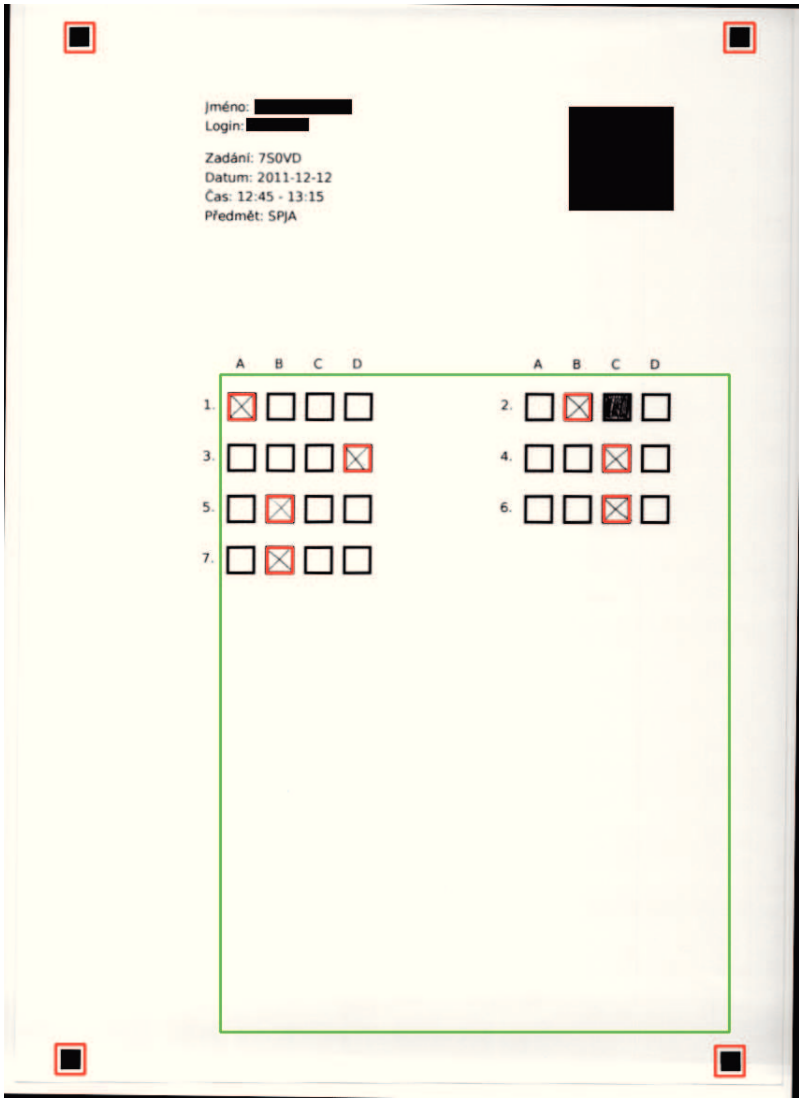
Tabulka 4: Výsledky detektoru při vzájemném porovnání boxů

Výsledky byly velice dobré, ale jako v předchozím případě, kvůli malému počtu dat je nelze považovat za objektivní a bylo třeba ještě provést testy na větším množství dat. Tresh level byl určen experimentálně a optimalizován na základě testovacích dat.

Počet otázek	406
Počet správně detekovaných odpovědí	376
Počet špatně detekovaných odpovědí	30

Tabulka 5: Výsledky evaluace nad větším množstvím dat

Z tabulky 5 vidíme, že úspěšnost detektoru při testování většího množství dat (58 testů, každý 7 otázek) je 92.6 %. Lze tedy říci, že navržená aplikace funguje správně a je připravena na reálné nasazení.



Obrázek 24: Ukázka výstupu detektoru

6 Závěr

Z testování detektoru plyne, že zpočátku detektor podával velice proměnlivé výsledky. Data z jednoho skeneru byl schopen úspěšně detekovat, ale jakmile dostal data z jiného skeneru, tak selhal. Postupnými úpravami se podařilo najít univerzální nastavení, které dosahovalo vysokých procent úspěšnosti i když data byly z různých skenerů. Poslední test, který byl proveden nad více než 50ti testy dosáhl výsledku kolem 90% úspěšnosti. Tento výsledek lze považovat za velice dobrý a lze prohlásit, že detektor je připraven na reálné nasazení.

Implementace webu také proběhla bez větších problémů. Podařil se splnit jak požadavek na přehlednost webového rozhraní, tak i integrace se službami VŠB (přihlášení LDAP). Web byl navíc rozšířen o prvky, které nebyly součástí zadání této práce - interaktivní rozhraní, pro nahlášení chybné detekce studenty a generátor pro vytváření testů pro učitele. Při testování webu se nenarazilo na žádné problémy.

Souhrnně lze říci, že všechny cíle práce byly splněny a aplikace může být nasazena v reálném využití.

Rozšíření detektoru by například mohlo spočívat v rozpoznávání nejen čtvercových, ale i kruhových značek nebo implementace rozpoznávání zaškrtnutých odpovědí pomocí klasifikátoru (pattern matching rozpoznávání). Jelikož je aplikace nastavena univerzálně, co se týče detekce a tudíž u některých skenerů může být problém s menší úspěšností detekce, a proto je v některých případech nutné změnit nastavení aplikace, tak by další možností při rozšiřování detektoru mohlo být naimplementování interaktivního rozhraní, pro změnu nastavení detektoru (nastavení limitů pro evaluaci nebo nastavení maximální vzdálenosti nalezené přímky od referenční) namísto současného systému, kdy se změna nastavení provádí změnou údajů v konfiguračním souboru.

7 Reference

[1] TVRDÍKOVÁ, Milena. IT podpora správy dokumentů. In: [online]. [2012-02-22]. Dostupné z: http://www.systemonline.cz/clanky/it-podpora-spravy-dokumentu.htm

[2] Vytěžování dat z formulářů. In: [online]. [2012-02-22]. Dostupné z: http://www.profiscan.cz/vytezovani-dat.php

[3] ORCHARD, Kevin. The use of optical mark reading (OMR) for census data collection [online]. 1998. Dostupné z: http://www.ancsdaap.org/cencon98/ /papers/drs/drskevin.pdf

[4] Remark Office OMR 8. In: [online]. [2012-03-13]. Dostupné z: http://www.acrea.cz/files/programy/gravic/gravic-remark-office.pdf

[5] *Remark Office OMR* [online]. [2012-03-13]. Dostupné z: http://www.gravic.com/remark/

[6] *Addmen OMR Pro* [online]. [2012-03-14]. Dostupné z: http://www.admengroup.com/OMR_Sheet.htm#omr_sheet_download

[7] *Vision OMR software* [online]. [2012-03-14]. Dostupné z: http://www.omrsolutions.com/omr_icr_desgin_softwares/indiaoutsources/ /vision_omr/index.php

[8] *TeleForm* [online]. [2012-03-14]. Dostupné z: http://quexf.sourceforge.net/

[9] *QueXF* [online]. [2012-03-14]. Dostupné z: http://quexf.sourceforge.net/

[10] *Shared Questionnaire System* [online]. [2012-03-14]. Dostupné z: http://dev.sqs2.net/projects/show/sqs

[11] *OpenCV Wiki* [online]. [2012-03-10]. Dostupné z: http://opencv.willowgarage.com/wiki/

[12] Gimp - vylepšení fotografií (30.). [online]. [2012-03-08]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=884

[13] SOJKA, Eduard. VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA OSTRAVA, Fakulta elektrotechniky a informatiky. *Digitální zpracování a analýza obrazu* [online]. Ostrava, 2000. ISBN 80-7078-746-5. Dostupné z: http://mrl.cs.vsb.cz/people/sojka/dzo/digitalni_zpracovani_obrazu.pdf

[14] HÝNA, Petr. *Detekce rohů v obraze* [online]. Brno, 2007. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=4930. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Michal Španěl.

[15] KANĚČKA, Petr. *Vyhledání význačných bodů v rastrovém obraze* [online]. Brno, 2007. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=1354. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Adam Herout, Ph.D.

[16] Moravec Operator. [online]. Dostupné z: http://kiwi.cs.dal.ca//d̃parks/ /CornerDetection/moravec.htm

[17] AMBROŽ, Ondřej. *Rekonstrukce 3D scény z obrazových dat* [online]. Brno, 2010. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=9243. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Michal Španěl.

[18] MOTÁČEK, Vladimír. *Panoramatické snímky automaticky* [online]. Brno, 2008. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=3520. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Vítězslav Beran.

[19] Implementace detekce hran v obrazu na vícejádrovém signálovém procesoru. [online]. [2012-03-09]. Dostupné z: http://www.comtel.cz/files/ /download.php?id=5466

[20] JANDA, Miloš. *Cannyho operátor a další používané hranové detektory* [online]. Brno, 2008. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=6269. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Jiří Venera.

[21] LEIKEP, Bořek. *Houghova transformace pro detekci čar* [online]. Brno, 2009. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=8869. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Michal Španěl.

[22] Hledání parametrického popisu objektů pomocí Houghovy transformace. [online]. [2012-03-05] Dostupné z: http://cmp.felk.cvut.cz/cmp/courses/ZS1/ /Cviceni/cv5/hough.htm

[23] Project 4: Feature detection. [online]. [2012-03-05]. Dostupné z: http://www.sci.utah.edu/čscheid/spr05/imageprocessing/project4/

[24] BOREK, Bernard. Úvod do architektury MVC. [online]. [2012-03-16]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[25] VONDRÁK, Ivo. VYSOKÁ ŠKOLA BÁŇSKA – TECHNICKÁ UNIVERZITA OSTRAVA, Fakulta elektrotechniky a informatiky. *Úvod do softwarového inženýrství* [online]. Ostrava, 2002. Dostupné z: http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf